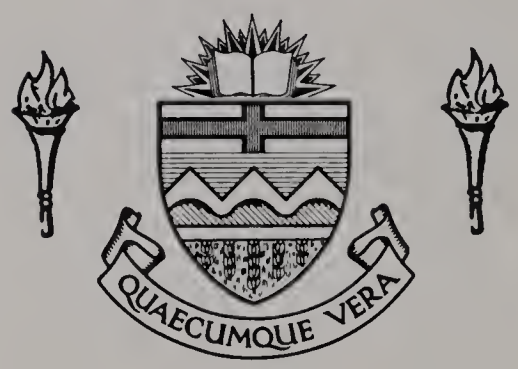


T H E U N I V E R S I T Y O F A L B E R T A

RELEASE FORM

NAME OF AUTHOR: Alan R. Covington

TITLE OF THESIS: Organization and Representation of Knowledge in a Semantic Net

DEGREE FOR WHICH THESIS WAS PRESENTED: Master of Science

YEAR THIS DEGREE GRANTED: 1980

THE UNIVERSITY OF ALBERTA

ORGANIZATION AND REPRESENTATION OF KNOWLEDGE IN A SEMANTIC NET

by

Alan R. Covington

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA

SPRING 1980

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "Organization and Representation of Knowledge in a Semantic Net", submitted by Alan R. Covington in partial fulfilment of the requirements for the degree of Master of Science.

Abstract

Various conceptions of the world generated by modal predications, such as "John believes that ... ", can be represented in semantic nets. A method to organize the propositions embedded within such conceptions is proposed. It uses a subnet to hold the access links to the propositions relevant to a particular view of the world. Propositions are organized within the net that references them, thus ensuring that different conceptions of the world are separated. "Virtual concepts" are introduced to provide the attachment points for some of the organizational structures within a subnet. Another organizational structure associated with each net is a concept access skeleton which provides access to entities via their known types.

A sub-problem of the Symbol-Mapping problem is discussed. The sub-problem is that of inheriting properties and relationships from a dependent superordinate concept to a corresponding dependent subordinate concept (e.g., relationships among parts of the generic elephant to the corresponding parts of a particular elephant). A partial solution employing function tables is described.

Acknowledgements

I am grateful to Len Schubert for his patience and guidance. He provided invaluable insight which put me back on the correct track many times.

I appreciate the useful comments on style and content that my committee, Louis Marckworth, Jeff Sampson, and Kelly Wilson, made.

Special thanks must go to Jim Lamont who paved the way by going first. He carefully read a very rough draft, and his comments on style and clarity saved me much time.

Lisa Higham and Chris Gray provided moral support and ears willing to listen to some of my crazier ideas.

Finally, thanks must go to Randy Goebel who did the research upon which this thesis is based. His enthusiasm and support helped make this thesis possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

INTRODUCTION

1.1 Outline

This thesis concerns the organization and representation of knowledge that forms various conceptions of the world, and to a lesser extent, property and relationship inheritance from dependent superordinate to corresponding dependent subordinate concepts, in a semantic net. The work builds on Goebel's <1977> thesis and consists of several parts which are outlined below.

Conceptions of the world are formed by some modal predicates such as propositional attitudes. For example, the sub-propositions enclosed by the modal predication "John believes that ... " form a conception of the world of John's beliefs. Most network theorists (e.g., Shapiro <1971>, Schank <1973>, Rumelhart and Norman <1973>, Bobrow and Winograd <1977>, Goebel <1977>, Fikes and Hendrix <1977>, Hendrix <1975,1979>) can represent modal predications, but no organization has been proposed for efficiently accessing the sub-propositions contained within such predications. Such an organization is important so that questions such as "What colour does John believe elephants to be?" can be as easily answered as "What colour are elephants?". The proposed organization which allows this ease of access

separates knowledge inside a particular conception of the world into a subnet of its own. Within this subnet knowledge is organized in the same fashion as in the main net. This organization requires a change in the internal representation of concepts since a concept can appear in many different subnets. In the new representation, a virtual concept appears in each subnet in which that concept is referenced. Virtual concepts provide the attachment points for the access structures which organize the propositions referenced in that subnet. Virtual concepts are not used as constituents of propositions. Instead, all propositions are constructed from concept nodes in the main net. This arrangement has the advantage that the system need have only one copy of any proposition or sub-proposition in the Knowledge base, thus reducing storage costs and facilitating proposition matching across subnets (e.g., determing whether John's beliefs about cats are true requires matching John's beliefs against the system beliefs.) Hendrix's <1979> "partitions" bear some resemblance to the present subnets, since modally embedded sub-propositions are placed in partitions separated from the main net. However, related sub-propositions (e.g., those belonging to a particular individual's set of propositional attitudes, or to a particular story - see section 2.4) are not necessarily grouped into one partition, nor are they organized for efficient access.

Another organizational structure associated with each

net is a concept access skeleton which provides access to entities via their known types. This is particularly useful in subnets where an explicit concept hierarchy is often not present. For example, in the story of "Little Red Riding Hood", the facts that the wolf is an animal and the grandmother is human are not mentioned. The listener is supposed to make these assumptions from his "real world" knowledge. When a question, such as "What animals are in the story?", is asked, this "real world" knowledge is used to answer it. It is possible to answer this question without using a concept access skeleton, but this requires searching the explicit concept hierarchy which is usually partly in the "real world" and partly in the subnet. The search may be time consuming, since many irrelevant branches may be examined. The concept access skeleton associated with a subnet can greatly reduce the search time because it is the minimum tree structure required to classify the concepts in the subnet, and because the individuals classified under a given type can be accessed by direct descent in this tree structure without trial-and-error search. For example, to find all the animals in a subnet, only those branches of its concept access skeleton which lead to instances of animals need to be traversed; this portion of the access skeleton in general corresponds to a small fragment of the main concept hierarchy.

One important question not examined by this thesis is question of the use and meaning of modal predications. This

question is being explored by Moore <1977>, McCarthy <1977> and Creary <1979> among others, but an answer to it will not be of much use without an efficient means of accessing the relevant knowledge, such as proposed in this thesis.

Property and relationship inheritance is another area examined by this thesis. The requirement for property and relationship inheritance arises when knowledge is not stored with a concept but is generalized and stored with a more general concept. For example, the knowledge that robin heads are joined to robin necks would not be stored with the 'robin' concept but would be generalized to, say, bird heads are joined to bird necks, and stored with the more general concept 'bird'. The particular sub-problem examined in this thesis is the one of inheriting properties and relationships between corresponding concepts dependent on concepts that are in a generalization relationship (e.g., from bird head to robin head). To facilitate property and relationship inheritance between corresponding dependent concepts, "Skolem function tables" are employed. The idea of the function tables is that corresponding dependent concepts (e.g., robin head and bird head) are accessed from their "supporting" concepts (e.g., robin and bird) by the same function name (e.g., 'h(x)' and 'h(y)' represent the concepts robin head and bird head respectively where 'x' is a robin and 'y' is a bird). Function tables facilitate the inheritance of knowledge because matching of corresponding dependent concepts becomes merely a table look up rather

than a search. This method of establishing correspondences is more direct and logically better founded than any method previously proposed. How the Skolem functions for the robin's head and the bird's head acquire a common name is not discussed in this thesis, but a general solution based on description matching appears to be feasible <Schubert 1979>.

Some work has been done on Goebel's <1977> access structure to improve its performance. A new algorithm for building and using his access structure is introduced. It has significant savings in time and space over Goebel's <1977> proposal and savings in time over Schubert et al.'s <1979> proposal when not all branches of the access structure are used.

The representation of credibility does not follow Goebel's <1977> proposal. Instead, it follows the system of Schubert et al. <1979>, but uses a relational rather than a functional notation.

Goebel's <1977> implementation of the knowledge base system has been rewritten to accommodate the changes and new ideas proposed in this thesis and to make it easily extensible for future work. A simple query language and answerer is included as part of the implementation. The query answerer demonstrates the use of the organization of different conceptions of the world, concept access skeletons and Skolem function tables.

In summary, the following has been accomplished in this thesis. An organization for efficiently accessing modally embedded propositions has been developed. A structure to quickly retrieve a concept via its known types or via super-types of its known types has been designed. This concept access structure works with the subnet proposal so that concepts are retrieved first by subnet and then by type. A method for efficiently rematching corresponding dependent concepts to facilitate property inheritance has been proposed. Finally, some performance improvements have been made to Goebel's <1977> and Schubert et al.'s <1979> topic access structure. These accomplishments are steps towards the develpment of an intelligent, domain-independent conversational system. An essential requirement for such a system is that it be able to perform the kinds of inference needed to support language understanding, consistency checking and question answering as effortlessly as people, in a manner which is more or less independent of the size of the available knowledge base. This in turn calls for an organization permitting highly selective, efficient access just to the knowledge relevant to a particular statement, question or goal within an arbitrarily large knowledge base. Without such an organization, even the simplest inference tasks would be combinatorially explosive.

1.2 Organizational Overview

The knowledge representation and notation used in this paper is presented in chapter 2. Chapter 2 also contains a description of the proposed method for representing different conceptions of the world and a discussion of the removal of "duplicate" sentential forms. Chapter 3 discusses two structures for organizing knowledge within a net. They provide topical access to propositions via the concepts involved in the propositions <Goebel 1977, Schubert et al. 1979> and to concepts via the known properties of the concepts. Both structures are represented internally in the same fashion and are joined together to form one super-structure. Chapter 4 discusses the procedures by which knowledge is attached to the appropriate points in this super-structure. Chapter 5 describes the sub-problem of the property and relationship inheritance problem examined in this thesis and the solution obtained. The data structures used to implement the knowledge representation and access structures are described in chapter 6. Chapter 6 also contains a description of the implemented query language and query answerer. Chapter 7 contains conclusions and indicates areas of further research suggested by this thesis.

Chapter 2

KNOWLEDGE REPRESENTATION

Semantic networks have been favoured by many people for the represention of declarative knowledge (e.g., Quillian <1968>, Shapiro <1971>, Schank and Rieger <1974>, Schubert et al. <1975,1979>, Hendrix <1977,1979>). The particular form used in this paper derives from that developed by Schubert <1975>, Cercone <1975> and Goebel <1977>. The structure has been altered to handle different conceptions of the world. Schubert <1975> noted that predicate calculus subsumed the network notations current at the time in expressive power. He designed a graphic notation that is a near isomorphism to predicate calculus. This graphic analog makes the transformation from predicate calculus to computer data structures mechanical and relatively easy.

This chapter is divided into five subsections. The first briefly describes the predicate calculus notation and its graphic analog used throughout this paper. The second describes a credibility scheme. The third deals with the problem of normalization. The fourth describes the alterations made to accommodate different conceptions of the world. The fifth deals with "duplication" of sentential forms.

2.1 Predicate Calculus Representation

Instead of using the standard predicate calculus notation of predicate followed by arguments, the infix notation described in Goebel <1977> is used. In this notation the second concept mentioned is denoted as the predicate. Table 2.1 gives a few examples of the correspondence between standard predicate calculus notation and the infix notation.

Prefix notation	Infix notation
P(x)	[x P]
Likes(Tom,Alice)	[Tom Likes Alice]
Gives(Tom,Alice,book)	[Tom Gives Alice book]

Table 2.1 Correspondence between Prefix and Infix Notations

Predicate and individual concepts are not syntactically distinguished internally, since a concept first encountered in the role of a predicate argument may later appear as a predicate or vice versa. For example, assuming that nothing is "known" about 'grey' and 'colour', consider the propositions

[grey colour]

[Clyde grey].

If they were entered in the above order and an internal syntactic distinction were made between predicates and non-predicates then the second proposition would result in an

error. As was mentioned above, a predicate-argument distinction is made in the surface text, the second top level constituent of the proposition always being the predicate. All concepts have internal (logical) names. The internal name for a concept is a tag which names the concept's address. The names in the surface text used in this thesis are assumed to be internal names except for variables. Variables are given unique internal names.

The graphic analog has a small circle for each proposition and a larger circle or oval for each concept. As is standard in semantic networks, a named concept appears only once. A concept may be named or unnamed. An unnamed concept is one whose internal name is unimportant to the understanding of the proposition (on paper). The proposition circle has arcs pointing to the various constituents of the proposition. The arcs are named to indicate what role[1] the particular part plays in the proposition (e.g., predicate, argument 1, argument 2, etc.). For example the third proposition of table 2.1 translates to the graphic representation in figure 2.1.

Predicates which take propositions as arguments, such as 'believes' and 'causes', and logical connectives are used

---

[1] Case rules, class requirements etc. are not recognized since the surface text is assumed to come from a natural language parser that "understood" the sentence parsed and thus will have taken this sort of requirement into consideration.

to build larger propositions from atomic propositions. The logical connectives represented in the current system are 'and'/&, 'or'/|, 'implies'/=>, 'not'/¬ and 'necessarily'/¤. In the graphic representation arcs that point to the sub-propositions of a logical proposition are dashed lines. In the graphical notation 'and' and 'or' are n-ary connectives (n≥1); 'implies' is dyadic; and 'not' and 'necessarily' are monadic.

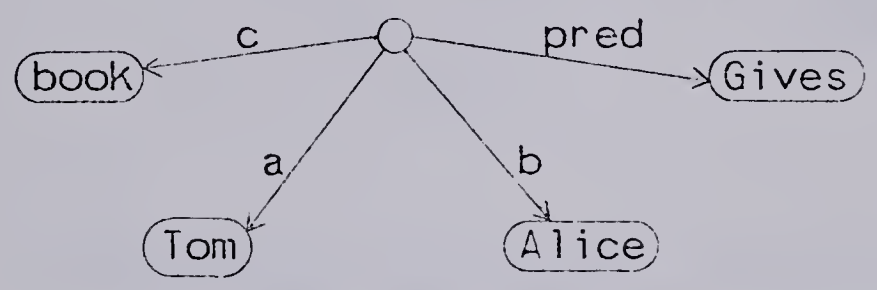
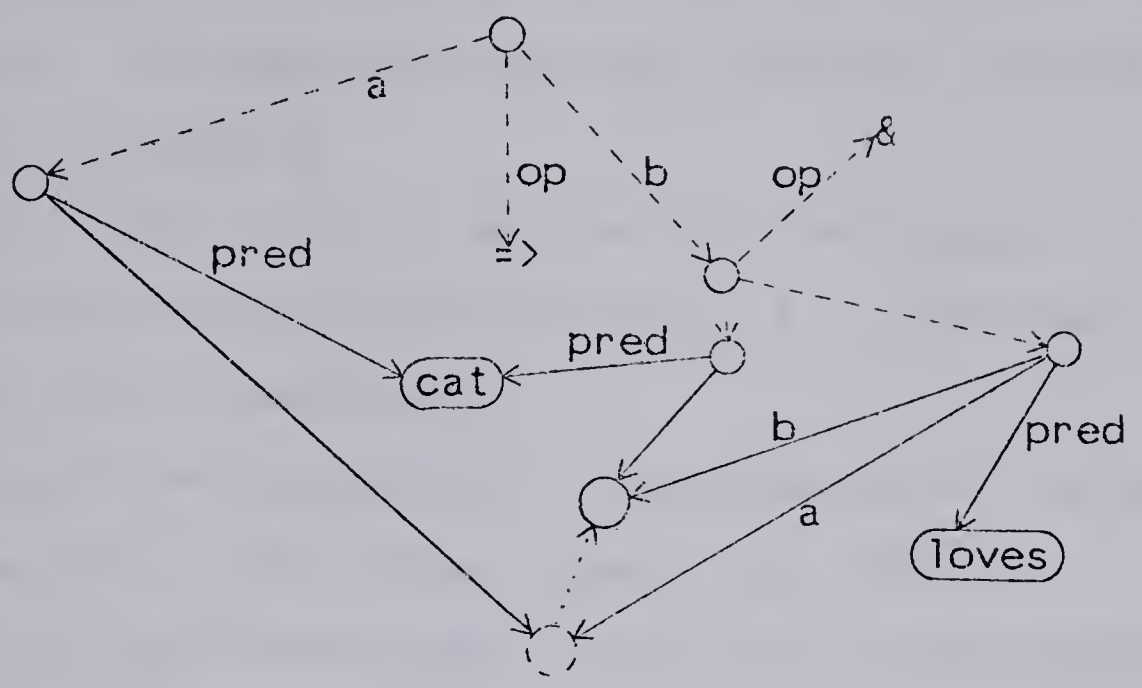


Figure 2.1 Tom gives Alice the book

To maintain the same expressive power as predicate calculus, quantification and scope inclusion or Skolem functions must be represented. The infix notation uses the standard ∀ and ∃ notation for quantification and order of variable binding for scope inclusion.

The graphic representation uses a circle with a broken edge to represent a universally quantified concept and a circle or an oval with a solid edge for an existentially quantified concept. Scope inclusion is represented by a dotted arc from the includer to its dependent. An example of quantification, scope inclusion and logical connectives

in the infix notation and its graphic analog is given in figure 2.2.



∀x∃y[[x cat]=>[[y cat]&[x loves y]]]
"All cats love some cat."

Figure 2.2 Quantification, Scope Inclusion and Logical Connectives

Modal prediates, such as 'believes' and 'hopes', can create opaque contexts <Quine 1971a,b, Hintikka 1971>. For example, the sentence "Mary wants to marry a millionaire." provides an opaque context for "a millionaire", if Mary is not supposed to have any specific, real-world millionaire in mind. On the other hand, the sentence "Mary wants to marry a millionaire she met yesterday." provides a transparent context for "a millionaire she met yesterday", since it refers to an actual person and any other reference to that person can be substituted for the phrase "a millionaire she met yesterday" without altering the meaning of the sentence.

The logical difference between these two sentences is that between quantification within the scope of the modal predicate and quantification into the scope of the modal predicate. The above two sentences would be represented in predicate calculus as

[Mary wants ∃x[[Mary marries x]&[x millionaire]]]

∃x[[Mary wants [Mary marries x]] & [x millionaire] & [Mary meets<yesterday> x]].

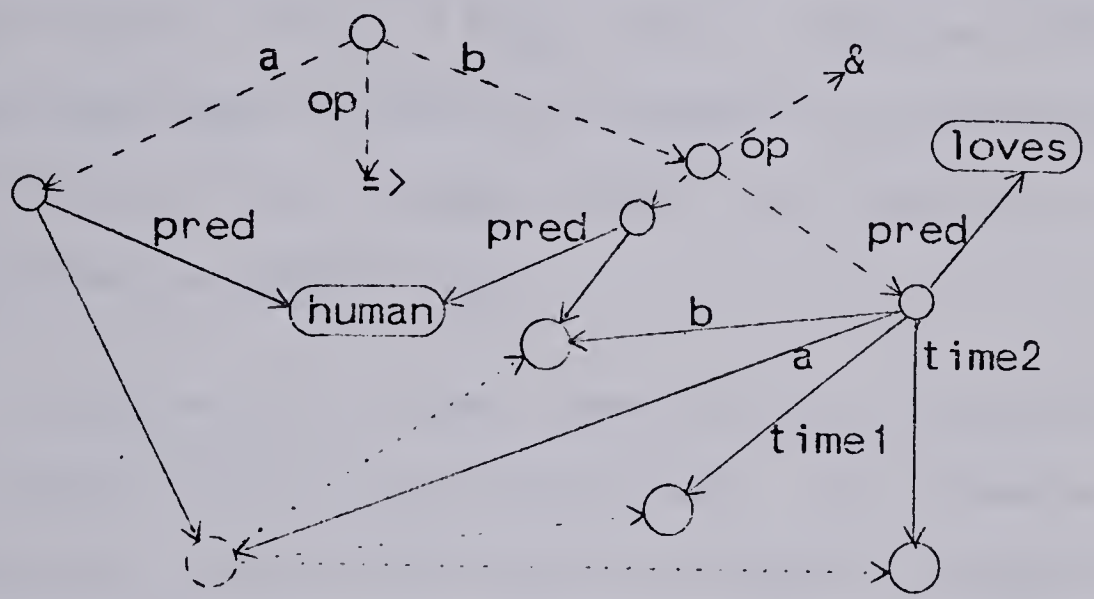
To represent the dependency of variables within the scope of an opaque modal predication, scope links from the proposition node of the modal predication to the dependent variable are used.

In the network notation of Schubert <1975> time is given a special role. All propositions hold for all time, but the time arguments give the time at which the predicate applies to the non-time arguments. With no time arguments, the proposition is timeless (i.e. the predicate applies to its arguments for all time); with one time argument, the predicate applies to its non-time arguments for an instant of time; and with two time arguments, the predicate applies to its non-time arguments for an interval of time. Time arguments are concepts so relationships can be built between them using time predicates such as 'after' and 'during' [1]. The notation for time is as follows. In the written form

---

[1] For a more complete description of this system see Schubert <1975> and Cercone and Schubert <1975>.

time arguments are placed in angle brackets directly following the predicate. In the graphical form they are connected to the proposition circle by arcs labelled as time arguments. An example of time arguments is shown in the interpretation of "Everybody loves somebody for sometime." in figure 2.3.



$$\forall x \exists y \exists t \exists u [[x\ \text{human}] \Rightarrow [[y\ \text{human}] \& [x\ \text{loves}\langle t,u\rangle\ y]]]$$

"Everybody loves somebody for sometime."

Figure 2.3 Time Arguments

Functions are denoted by a left parenthesis followed by the function name, its arguments and a right parenthesis. In the graphical form a function is either represented by the concept to which it evaluates or as an explicit function node. A function node is a circle with labelled arcs indicating the function name and its arguments. The function node is used as the value of the function.

2.2 Credibility

A system that is to represent human knowledge must be able to express how credible it believes a proposition to be. Credibilities are not interpreted as truth values or as possibility values <Zadeh 1975>, but as degrees of subjective probablity <Schubert et al. 1979>. In general the present systems for representing credibility either use single numbers (e.g., Kling <1973>, Lefaivre <1974,1976>, Schank and Rieger <1974>) or probability distributions over truth values (e.g., Goebel <1977>) to indicate a proposition's credibility.

A problem with these forms is that they directly attach the credibility to the proposition. This does not allow the proposition to be used as part of another proposition. For example, if the system believes that dolphins are intelligent with credibility .9 then this proposition cannot be used when the system is told John believes that dolphins are intelligent with degree of belief .8. This inablity to use propositions or sub-propositions more than once results in storage waste. Another problem is that the form of credibility used does not allow comparisons between unknown credibilities. This occurs because the credibilities are directly part of the proposition, thus not accessible to be used as arguments and they must be given as a numeral or as an absolute distribution. Schubert et al. <1979> solve these problems by expressing credibility explicitly as a

function of a proposition. An equivalent approach taken by this thesis is to use the relational form rather than the functional form to represent credibility. The credibility of a proposition is given by a credibility proposition with a proposition and its credibility as arguments. For example, the above system credibility about the intelligence of dolphins is represented as

[∀x[[x dolphin]=>[x intelligent]] has-credibility .9].

For propositional attitude propositions, a relational rather than functional notation is also used to represent strength of belief. This is accomplished by extending propositional predicates from binary to ternary predicates. The third argument is the strength (credibility, degree-of-belief) the first argument has in the propositional argument (second argument). For example, John's belief about dolphin intelligence is expressed as

∀x[John believes [[x dolphin]=>[x intelligent]] .8].

Note that the sub-proposition in the above proposition is identical to the one in the system's credibility statement about the intelligence of dophins. Thus this sub-proposition can be the same one in both propositions.

The credibilities or strengths described here and in Schubert et al. <1979> are not restricted to numbers, so some partially or well ordered set of concepts other than numbers can be used to represent credibility. For example, the set {false, ..., very very unlikely, ..., possibly, ...,

likely, ..., very very likely, ..., true} could be used. Another advantage is that unknown credibilities or degrees of belief can be represented. For example, it might be known that John believes dolphins are intelligent but the strength of his belief is unknown.

2.3 Normalization

There are several advantages to having a fixed logical form rather than several logical forms or an arbitrary logical form. They are simplification of the classification scheme (see chapter 4), inferencing routines and pattern matching routines.

Schubert et al. <1979> proposed an antecedent-consequent form for propositions in order to support their "variable-sharing" scheme. The "variable-sharing" scheme has since proved to be untenable (see section 5.2). With this scheme abandoned, conjunctive normal form (CNF) was chosen as a suitable form for representing propositions since with CNF a proposition will appear in only one form in the knowledge base. Such is not the case with the antecedent-consequent form. Converting to CNF also aids in reducing the amount of duplicate knowledge in the system and aids in eliminating duplicate sentential forms. This is discussed further in section 2.5.

The graphic notation developed by Schubert <1975> uses scope inclusion arcs from universal variables to dependent

existential variables to indicate order of binding of variables. As part of the solution to the property and relationship inheritance problem (chapter 5), these scope inclusion arcs have been replaced by Skolem function tables. Skolem function tables are attached to universal variables. An existential variable, E, dependent on a universal variable, U, is indexed in the function table attached to U by a function name. The function name used is one that is generated when the proposition containing E and U is converted to CNF. For example, the proposition

∀x∃y[[x dog]=>[[y cat]&[x chases y]]]

is converted to CNF to yield

[[¬[x dog]]|[(f x) cat]]

[[¬[x dog]]|[x chases (f x)]]

where 'f' is the generated Skolem function name. In the Skolem function table attached to 'x', 'y' is indexed under the function name 'f'. Skolem function tables are equivalent to scope inclusion arcs with function names associated with the arcs. Skolem function tables only replace the scope inclusion arcs from universal variables to existential variables. They do not replace scope inclusion arcs that go from a proposition to a concept.

Modal operators and predicates can create opaque contexts, so that it may be impossible to convert a proposition containing a modal operator or predicate to CNF. For example, the two propositions

[John believes ∃x[x witch]]

$\exists x$[John believes [x witch]]

mean quite different things. The first says that John believes that there is a witch; the second says that John believes that someone in particular is a witch. The only difference between the two propositions is the scope of the existential quantifier. To solve this problem propositions are converted to modal conjunctive normal form (MCNF) <Hughes and Cresswell 1974>. MCNF is the same as CNF except that sub-propositions which are operands of modal operators are converted to MCNF as if they were top level propositions. If the normalized sub-proposition has a conjunction as its top level operator then the modal predication is distributed over the conjunction. The quantifiers dependent on the modal predication, (i.e. inside the scope of the modal predication), are not brought through the modal predication but remain inside its scope. Thus the correct sense of the proposition is maintained. For example, the proposition

[John believes $\forall x \exists y$[[x cat]=>[[y dog]&[x chases y]]]]

when converted to MCNF becomes

[John believes $\forall x$[[$\neg$[x cat]]|[(g x) dog]]]

[John believes $\forall x$[[$\neg$[x cat]]|[x chases (g x)]]].

2.4 Conceptions of the World

Modal predicates allow the system to define conceptions of the world in which it can express propositions contrary to the main world knowledge without contradicting the main world knowledge. The modal predicates dealt with in this thesis are ones that take a concept as first argument and a proposition as second argument. This class of predicates includes mental attitudes (e.g., belief and hope) and story predicates (e.g., is-a-story-in-which and is-a-poem-in-which). These predicates associate with their first argument a conception of a world. For example, the sub-propositions enclosed by the modal predication "John believes ... " form a conception of the world of John's beliefs. Stories form another type of conception of a

```
ǂx[x is-a-story-in-which ǂyǂw[[y pumpkin]&[w girl]&
... ]]
```

Figure 2.4 Story of Cinderella

world, namely the world of the story. The idea of a story as a concept is well established in English dialogue. For example, in talking about "Cinderella" one might say, "This story has a pumpkin which turns into a coach." or "The story's title is 'Cinderella'.". A fragment of the story of Cinderella is shown in figure 2.4.

Since the sub-propositions of a modal predication form

a conception of the world, it seems reasonable to separate these sub-propositions into bundles, each bundle containing the sub-propositions pertinent to a particular conception of the world. This is done by putting the sub-propositions into a subnet. The problem of organizing the sub-propositions within a subnet is solved by applying the organizational methods of the main net to the subnet. The modal super-predications do not interfere with this organization since in the subnet the sub-propositions appear as top level propositions. For example, the sub-propositions,

[x unicorn]

[x white],

embedded in

[John believes ‡x[[x unicorn]&[x white]]]

are put into a subnet where they appear as if they are top level propositions. Thus, they can be organized in the subnet in the same fashion as propositions in the "real world", ignoring the fact that they are embedded in a modal predication.

There are two alternate methods for deciding what sub-propositions belong to what subnet. The first is to generate a new subnet for each modal-predicate/first-argument pair. Only the sub-propositions enclosed by a particular modal-predicate/first-argument pair are put into that pairs subnet. For example, a different subnet would be used to organize the sub-propositions embedded by the

predications [John believes ...], [John hopes ...], and [Mary pretends ...]. The second method is to generate only one subnet for the first argument. All sub-propositions of modal predications involving a particular concept as first argument would go into that concept's subnet. For example, all sub-propositions embedded in the predications [John believes ...], [John hopes ...], and [John desires ...] would be contained in one subnet, the subnet associated with John.

The first method would separate such statements as "John believes in unicorns." and "John hopes to win the race." into two separate subnets. The super predication "John believes" or "John hopes" can be removed using this scheme as it is implied by the particular subnet in which the sub-proposition resides. Several problems are created by this method. First, a lot of subnets would be generated. For one person there would be as many subnets as there are mental attitudes. This is quite space inefficient as each subnet has its own dictionary and concept access skeleton (see chapter 6). Second, the many subnets split up the conception of one person's mental world making it hard to find a particular attitude towards some subject. This would have to be accomplished by searching all the subnets associated with a person. Hendrix's <1977,1979> system has this problem since it splits a conception of the world into many partitions. Third, removing the super-proposition node removes the capability of stating a credibility proposition

about the super-proposition.

The second method does not have most of the above problems. The conception of a person's mental world is not split up as it is all in the one subnet attached to the concept representing that person. Since the sub-propositions in this one subnet could have many different modal super-predications, it is necessary to keep the modal super-predications. This solves the third problem because the top level proposition node is present for attachment of a credibility proposition. This method appears to be the best one of the two, and is the one which is implemented. The only problem which remains is one of space efficiency since the super proposition is kept.

Another problem introduced by propositional attitudes is that of identifying individuals between conceptual worlds; for example, identifying Mary in John's conceptual world with Mary in Tom's conceptual world. Hintikka <1971> does this in modal possible worlds with a set of partial functions F, such that if f ∈ F is applied to a possible world it returns at most one individual. The individuals obtained by applying the function to all worlds are manifestations or virtual individuals in the various worlds of one individual. In the system of conceptual worlds developed here, this function 'f' can be thought of as the logical name of the individual. Looking up a logical name in a particular subnet will retrieve the virtual concept

that corresponds to that logical name in that subnet. The existence in a conceptual world of a concept does not necessarily mean that a dictionary lookup for the concept in the corresponding subnet will succeed. For example, Mary could exist in John's conceptual world, but if the system does not know any of John's attitudes towards Mary, a virtual concept corresponding to Mary will not appear in John's subnet. Thus, dictionary lookup can only be used to identify virtual concepts between conceptual worlds. It cannot be used to determine the existence of a virtual concept in a conceptual world.

Virtual concepts are not used as propositional constituents since only one copy of a proposition or sub-proposition exists in the knowledge base (see section 2.5). A virtual concept provides the attachment points for the access structures which organize knowledge about the virtual concept. For example, if the two statements "Tom is a cat." and "John believes Tom is a dog." are entered into the knowledge base then the fact that Tom is a cat is indexed under the concept Tom in the main net and the fact that in John's subnet Tom is believed to be a dog is stored under the virtual Tom in that subnet. The same concept 'Tom' is used by both propositions. Virtual concepts allow knowledge to be accessible from the appropriate subnet without sacrificing the advantages of having only one copy of any proposition or sub-proposition in existence in the knowledge base (see section 2.5).

To show that the main net and subnets are structurally equivalent, a summary of how propositions, the main net and subnets fit together is presented below. The knowledge base consists of a set of propositions and a main net. The main net has a set of concepts which have links to the propositions that involve them. In addition each concept possibly has associated with it a subnet which holds its conception of the world. A subnet has a set of virtual concepts which have links to the propositions that involve them. Also each virtual concept possibly has associated with it a subnet which holds its conception of the world, as conceived of within the superordinate subnet. Thus the idea of a subnet is a recursive one. Subnets can be nested to any depth. For example, a nesting of depth two would be created by statements of the form "John believes that Mary believes ... ". The only seeming difference between the main net and subnets is that the main net has concepts that are used as constituents of propositions. This function can be logically removed from the main net concepts leaving the main net concepts only as attachment points for the various access structures (i.e. identical to the virtual concepts of subnets). The virtual concepts of the net in which the proposition is asserted are place holders for the concepts used as propositional constituents. If the main net and subnets are viewed in this fashion it becomes apparent that the main net and the subnets are identical in structure. In fact, the main net is the systems conception of the world.

2.5 Duplicate Sentential Forms

This section defines what is meant by "duplicate" sentential forms, gives motivation for avoiding them and describes the method by which they are removed in this system.

The definition of "duplicate" sentential forms is as follows. Let sf be a sentential form that is part of an input proposition, p, then the phrase "sf is duplicated" means that a sentential form in the Knowledge base, say SF, has the same constituents as sf. Thus sf can be replaced in proposition p by SF with no change in the meaning of p. A constituent of a sentential form is either a sentential form or a concept. To complete the definition of a duplicated sentential form, a definition of when concepts in two sentential forms are "the same" is required. This is not given here, but after non-duplication of sentential forms has been motivated, since the definition is quite complex and implicitly includes the method of their removal.

There are several reasons for having sentential forms appearing only once in the Knowledge base. Considerable space is saved since common sentential forms are not repeated. This is particularly evident when an implication with a large conjunction as consequent is entered. Once the proposition is put into MCNF the antecedent is repeated for each conjunct of the consequent. For example, consider the proposition

∀x[[x elephant]=>[[x mammal]&[x grey]].

Conversion to modal conjunctive normal form yields the two clauses

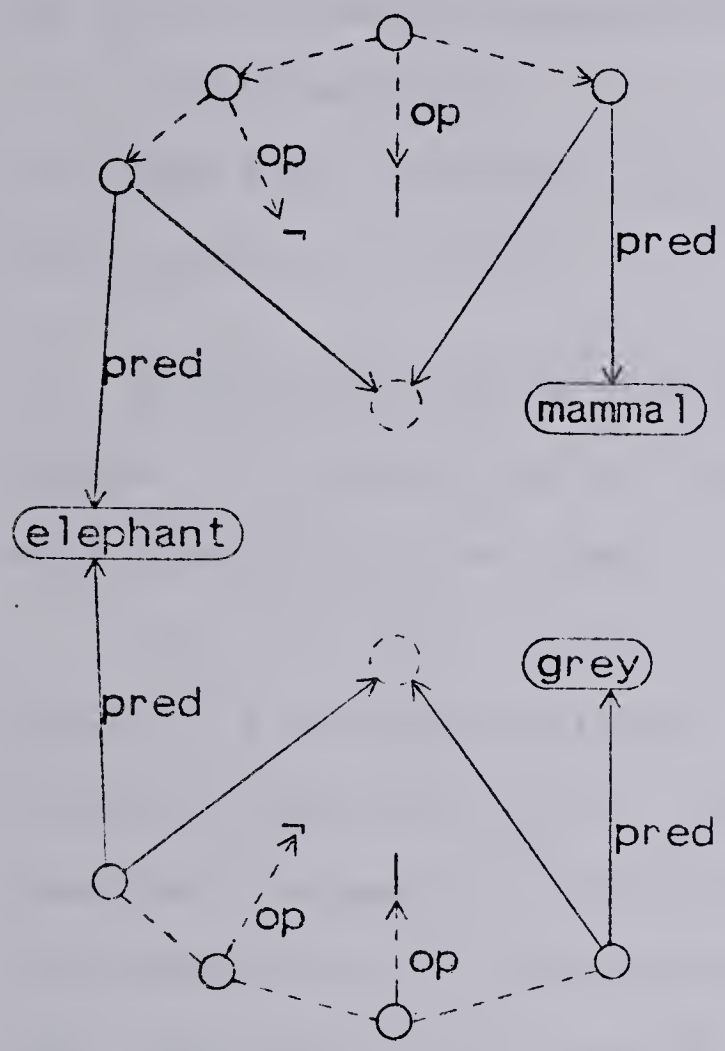
[[¬[x elephant]]|[x mammal]]

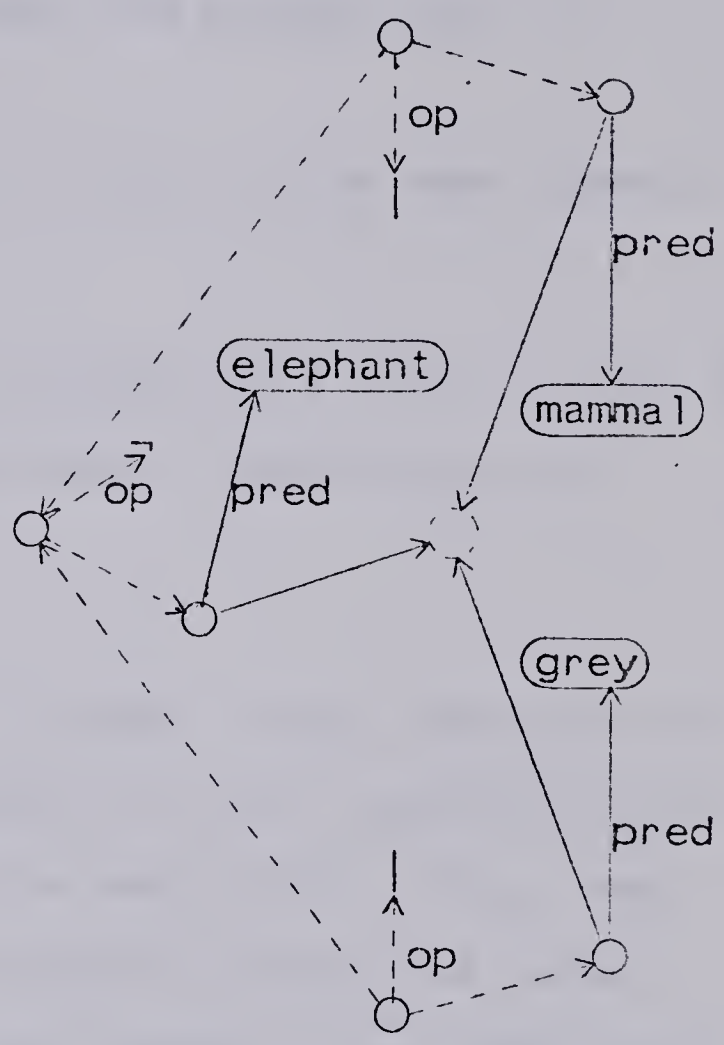
[[¬[x elephant]]|[x grey]].

Without sharing sentential forms, they are represented as in figure 2.5a; with shared sentential forms, they are represented as in figure 2.5b. Another reason for sharing sentential forms is that it facilitates inferencing since continual matching of universal arguments is not required. For example, as can be seen in figure 2.5b, only one universal elephant concept is present. Thus, if one wants to apply the mammal knowledge to elephants then the information "all elephants are mammals" is accessed and used only once. The single application links the two primary universal nodes of elephant and mammal. This is equivalent to doing the first step of unifying all elephant knowledge with all mammal knowledge at once. A further reason for sharing sentential forms is that matching propositions or sub-propositions across subnets, such as to compare John's explicit beliefs about cats to Henry's, is fast. Only proposition nodes need to be compared since different proposition nodes are guaranteed to represent different propositions. Finally, removal of explicit redundant knowledge occurs automatically since propositions only occur once.

Replacing sentential forms in the input by their

duplicates in the Knowledge base (if they have one) is accomplished in the following manner. First duplicate concepts must be replaced since they are the bottom level





∀x[[x elephant]=>[[x mammal]&[x grey]]]
in modal conjunctive normal form

Figure 2.5a Figure 2.5b
Non-shared and Shared Sentential Forms

constituents of sentential forms. There are three types of concepts that have to be considered. They are constants, universally quantified variables, and existentially quantified variables.

A constant, c, in the input proposition is duplicated

in the knowledge base if there exists a concept in the knowledge base with the same logical name as c's logical name. For example, if the proposition

[Clyde elephant] (1)

is in the knowledge base and then the proposition

[Tom elephant] (2)

is added then 'elephant' in (2) refers to the same concept as 'elephant' in (1).

Universally quantified variables are a little harder to handle. It seems that all statements containing such variables are of the form

∀x...[[[x P]&...]=>...]

where P is a type predicate. In other words, the universal claim is made only for all things of type P, and this type constraint appears as implicative antecedent. Thus, when the proposition is converted to modal conjunctive normal form the variable is used as the argument of a "negatively occurring" type predicate. A "negatively occurring" predicate is one that occurs in a negated atom when the proposition is normalized. A "type" predicate is similar to Moore's <1975> class concept. Every concept has at least one intrinsic "type" which cannot be negated without radically changing the concept. For example, if some x is assumed to be grey and an elephant, then a subsequent revision of this information is relatively minor if it involves replacement of [x grey] by [x white], but radical if it involves replacement of [x elephant] by [x reptile].

Some examples of type concepts are 'elephant', 'mammal' and 'pencil'. Some examples of non-type concepts are 'grey' and 'guitarist'. The negatively occurring type predicate, say P, is the subject or main focus of the clause. The universal argument represents it throughout the clause. For example, in the proposition

∀x[[x elephant]=>[[x grey]&[x mammal]]],

'x' appears as argument of the type predicate 'elephant' and represents "all elephants" throughout the proposition. When this proposition is converted to modal conjunctive normal form the two clauses

[[¬[x elephant]]|[x grey]]

[[¬[x elephant]]|[x mammal]]

are produced. Within these clauses, 'x' is the argument of the negatively occurring type predicate 'elephant'. Thus, the universal variable in the knowledge base that has been used as an argument of P is the concept that duplicates the universal variable in the input proposition. There are several cases that have to be examined because there is not always a one-to-one correspondence between universal variables and negatively occurring type predicates as presupposed by the preceding statement. The other two cases are one negatively occurring type predicate to many universal variables and many negatively occurring type predicates to one universal variable.

The first case occurs in propositions such as

∀x∀y[[[x elephant]&[y elephant]]=>[x likes y]].

A solution to this is to allow more than one universal variable to be associated with a given type predicate. The universal concepts associated with a type predicate are attached to each type predicate via a linked list. The first one on the list is the "main one". It is used if only one universal argument is used with the type predicate in a proposition.

The second case, the one of many negatively occurring type predicates to one universal variable, occurs in propositions such as

∀x[[x mammal]=>[¬[x reptile]]]. (3)

A method to handle this is to allow each negatively occurring type predicate its "turn" in determining the replacement for the universal variable. This requires repeating the proposition in the knowledge base as many times as there are negatively occurring type predicates using one universal variable. For example, assuming that c.1 and c.2 are the universal variables associated with 'mammal' and 'reptile' respectively, then the above proposition would be entered in the knowledge base as

[[¬[c.1 mammal]]|[¬[c.1 reptile]]]

[[¬[c.2 mammal]]|[¬[c.2 reptile]]].

This seems to be defeating some of the motivation of this section, but an argument can be made that this type of knowledge should not be represented as it is above. The archtypical examples of this sort of knowledge come from partitionings (e.g., animals into mammals, reptiles, fish,

etc.), so rather than represent the partitioning of a concept into n subordinate concepts by n(n-1)/2 statements about disjointness (e.g., as above) plus n statements about subordination (e.g., animals are mammals, fish, reptiles, etc.), a single partitioning relationship should be used <Schubert 1979>. A partitioning relationship saves space because the disjointness and subordination statements are combined into one statement. For example, the partitioning of animals would be represented as

[animal partitioned-by mammal reptile fish ...].

The last class of variables to be examined are existential concepts. Discovering if an existential variable in the input has a duplicate in the knowledge base is a more difficult problem than the analogous problems for constants and universal variables. A solution is not presented in this thesis, but it does seem feasible using description matching <Schubert 1979>. A brief description of how this matching may be accomplished is given in section 5.2.

Once the duplicate concepts in the input proposition have been replaced by their duplicates in the knowledge base all that remains to be done is to replace duplicated sentential forms. This requires that the sentential forms already in the knowledge base must be accessible. To this end a hash table containing all sentential forms present in the knowledge base is maintained. An input sentential form

is in the knowledge base if it can be found in the sentential form hash table. The replacement of sentential forms is done from the lowest level sentential forms up to the top level proposition. This ensures that any sentential forms of the input that are in the knowledge base are replaced and it allows the use of much simpler hashing and comparison routines. The hashing and comparison routines need only look one level down from the sentential form in question (i.e. at the immediate arguments of the sentential form) since if the sentential form is in the knowledge base so are its arguments and they will already have been replaced.

Chapter 3

NET ACCESS STRUCTURES AND METHODS

Representing knowledge adequately is useless without being able to access it in a reasonable amount of time. The organization presented in this chapter allows knowledge to be accessed via the concepts involved in it using a topical organization of that knowledge and individuals to be found by the properties known about them. The first two sections describe the topical organization <Goebel 1977, Schubert et al. 1979> and some improvements made to it. The next section motivates the associative access of individuals via the properties known about them. The data structure chosen for the associative access of individuals is the same as the one used for the topical organization. The last section shows that these two structures can be joined together without impairing the operation of either. In fact, a small improvement in the topical organization results from this union.

Some work on topical organization of knowledge has recently been done by Reder and Anderson <1979> and Rychener <1979>. Reder and Anderson's <1979> emphasis is on the psychological motivation for such an organization, but structurally their proposal is much cruder than Goebel's <1977> and Schubert et al.'s <1979>. Rychener <1979> uses a one level division of the knowledge stored about a concept. This is in effect a one level topic hierarchy, but again is

cruder than Goebel's <1977> and Schubert et al.'s <1979>.

3.1 Topic Hierarchy

This section describes topic predicates and topic hierarchies (TH) as Goebel <1977> first proposed them and some changes made to his original definitions by Schubert et al. <1979>. These changes facilitate topic access skeleton (TAS) building and accessing, and aid in the intuitive interpretation of the semantics of topic predicates and how they affect classification. One alteration to topic predicates is made to allow greater classification power when dealing with n-ary predicates (n>1). Also, Goebel's <1977> definition of a TH is modified slightly to increase the efficiency of building and accessing TASs.

3.1.1 Topic Predicates

Goebel's <1977> view of topic predicates is that they provide a means to separate propositions into different categories or topics. A topic predicate takes a proposition as an argument. The credibility[1] of the topic predication indicates how well the proposition argument fits into that topic.

---

[1] This credibility could be used by reasoning components of the system to decide how relevant a proposition is to a query.

To aid in the classification of propositions this definition was altered by Schubert et al. <1979>. A topic predicate is now viewed as a predication over a proposition and a concept in that proposition. For example, using Goebel's <1977> definition the proposition

[Clyde grey]

would result in the classification

[[Clyde grey] colouring].

This can lead to some confusion since the concept referred to by the classification is not mentioned. The proposition

[Clyde grey]

is certainly not a 'colouring' proposition about 'grey', rather it is an 'instance' proposition about 'grey' and a 'colouring' proposition about 'Clyde'. The present definition results in the classification

[[Clyde grey] colouring Clyde]

which can be read as "[Clyde grey] is a colouring-proposition about Clyde". The main advantage of this form is that it says what concept is referred to by the classification, whereas, the original form just said to what topic the proposition belonged.

To classify a proposition-concept pair where the concept appears in an argument position within the proposition, it is necessary to find the topic "indicated"

by the predicate of the proposition[1]. In the above example it is necessary to know the topic indicated by 'grey' when classifying

[Clyde grey]

with respect to 'Clyde'. This knowledge takes the form of an indicator link from the concept 'grey' to the 'colouring' topic. Basically the indicator link is a short form for the predication

[grey indicates colouring].

It is possible to have a concept indicate more than one topic. For example, the concept 'clear' indicates the topic 'translucency' and the topic 'colouring'. In this case 'clear' is more relevant to 'translucency' than to 'colouring'. Thus, some means of indicating the relevance of a concept to a topic is required. This is done by making the 'indicates' predicate into a ternary predicate. The third argument being the relevance of the concept to the topic. The above indicator links for 'clear' translate into the propositions

[clear indicates colouring r1] (1)

[clear indicates translucency r2]

[r1 less-than r2].

How well a proposition-concept pair fits into a particular topic is specified by the relevance of the indicator link

---

[1] For a more detailed description of classification see chapter 4.

used to classify that proposition-concept pair. For example, the proposition

[glass1 clear]

is classified as

[[glass1 clear] colouring glass1].

The relevance value of the classification proposition is the relevance of the indicator link from 'clear' to 'colouring', namely 'r1'.

This form of the 'indicates' predicate is fine for monadic predicates but for n-ary (n>1) predicates the indicator link used may depend on the argument position. For example, the proposition

[A part-of B]

says that B has parts and that A is a part, quite different interpretations depending on the argument. This proposition should be classified with respect to A and B under two different topics. Thus, the 'indicates' predicate needs to be a second order quaternary predicate taking a topic, a first order predicate, argument number and relevance as arguments. For example, proposition (1) would be changed to a form like

[colouring is-indicated-by clear 1 r1]

and the 'part-of' predicate may have the indicator predications

[subparts is-indicated-by part-of 2 r3] (2)

[superpart is-indicated-by part-of 1 r4].

Currently, for efficiency, indicator propositions are

converted to indicator links from concepts to topics. An indicator link includes argument number and relevance value as part of it. For example, if proposition (2) were instantiated, it would be converted to a link from the concept 'part-of' to the topic 'subparts'. The link would have associated with it an argument number of 2 and a relevance represented by the concept 'r3'. The indicator propositions do not appear explicitly in the knowledge base. They would have to if the system was to know about this aspect of its own internal structure.

3.1.2 Structure

A TH (topic hierarchy) is built by specifying sub/super topic relationships among the topic predicates. A sub/super topic specification is a proposition, although currently it is represented in a special form to allow rapid access to sub/super topics from a given topic concept. For example, to specify the appearance sub-hierarchy of figure 3.1, the equivalent of the propositions

[appearance super-topic-of form r1]

[appearance super-topic-of colouring r2]

[appearance super-topic-of translucency r3]

[appearance super-topic-of texture r4]

would be asserted. The third argument in a sub/super topic specification is the relevance of a topic to its sub/super topic. Again, as with the 'indicates' predicate, sub/super topic predications do not appear in the knowledge base.

These would have to be asserted in the net to allow self knowledge to the system.

A TH allows inheritance of topical classifications similar to the manner in which a concept hierarchy allows inheritance of type classifications <Goebel 1977>. For example, in the hierarchy fragment in figure 3.1 a proposition classified under 'colouring' is also an 'appearance' proposition and an 'external-quality' proposition.

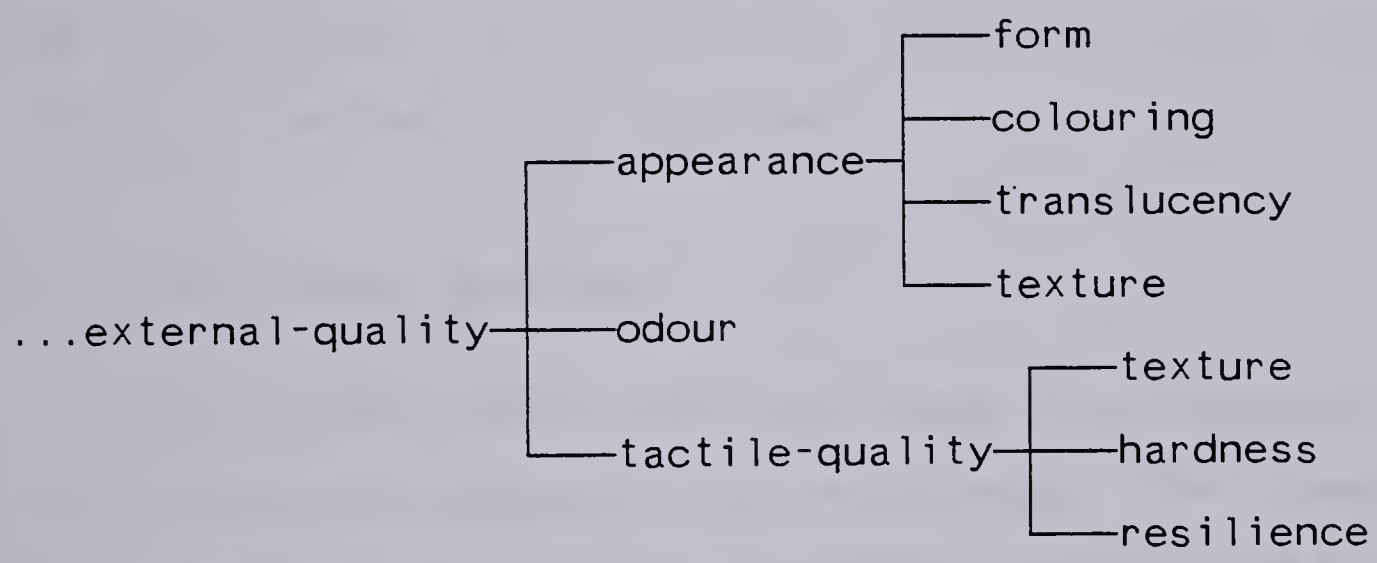


Figure 3.1 Topic Hierarchy Fragment <Schubert et al. 1979>

The change made to Goebel's <1977> definition of a TH is to restrict a TH to a tree. The motivation for requiring this is the resultant large reduction in the complexity of the TAS (topic access skeleton) building algorithm. This change is admissible since a TH apparently approaches a non-overlapping partitioning of predicates, for which the number of topics which appear twice in the TH should be small and

situated towards the leaf nodes of the TH. A topic that appears twice or more can be handled by making duplicate nodes, with different names, for that topic and having all the concepts that have a topic indicator link to that topic point to all the duplicate nodes. A sample TH of 53 nodes that Schubert et al. <1979> designed has only one node, a leaf node, used twice.

To change figure 3.1 to this new form, the 'texture' topic which appears twice would be represented by two topic, say 'texture1' and 'texture2'. Any concepts that previously had an indicator link to 'texture' would now have indicator links to 'texture1' and 'texture2'.

3.2 Topic Access Skeletons

TASs (topic access skeletons) have to be economical with respect to accessing time and storage. The storage must be minimized, since every constant concept and every existentially quantified variable has its own TAS associated with it. Time to access a particular node in a TAS must be minimized, since access to any proposition in the knowledge base is via a TAS node.

There are two basic types of TAS inquiries. One is for a particular node; the other is for a particular node and the TAS sub-structure attached to it. These two access types are not identical in the TAS structure proposed in this thesis and in the TAS structure proposed by Schubert et

al. <1979>. The differences are discussed in section 3.2.1. The first type of access is used to look for a particular proposition or for a group of closely topically related propositions (i.e. under one topic). The second type of access is used to access a group of loosely topically related propositions (i.e. under a sub-tree of the TH). This section discusses several different structures and methods of accessing TASs. It consists of three subsections. The first describes the structures and access methods, the second the timing costs, and the third storage costs for the various TASs. The methods discussed are the ones proposed by Goebel <1977>, by Schubert et al. <1979> and a new version which overcomes some of the problems of the previous two. Where appropriate they are compared against a linear list of backlinks from concept to propositions since this is the method the TAS supplants.

3.2.1 Structures and Access Methods

Goebel <1977> proposed to instantiate in a TAS only the branches of the TH that were used to classify knowledge under the concept associated with the TAS, thus eliminating the overhead of storing unused TAS branches. For example, if two propositions, say 'prop11' and 'prop265', were classified with respect to some concept under the 'hardness' and 'colouring' topics respectively then the TAS attached to that concept would appear as in figure 3.2. Assuming the TH as in figure 3.1 was used as the base for the TAS and that

no other propositions were classified under the concept.

The algorithm proposed to access a particular topic, A, in a TAS is as follows. One accesses the topic node of A in the TH and then ascends to the root of the TH keeping a list of the topic nodes one traverses in order of most recently traversed. This list is the path in the TH from the root topic to topic A. To access topic A in a particular TAS, one descends the TAS following the path of topic nodes found in the ascent of the TH. In the case of topic A not being present in the TAS, the descent of the TAS will come to a dead end. To access a sub-hierarchy rooted at A, exactly the same procedure as outlined above is followed.

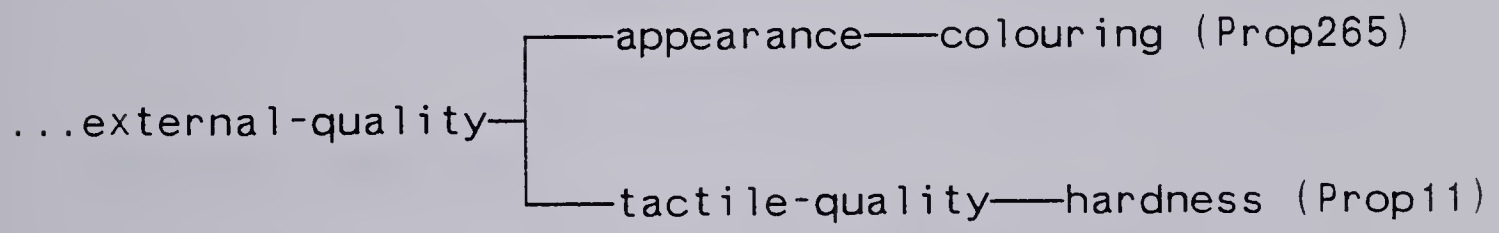


Figure 3.2 Topic Access Skeleton Fragment

Schubert et al. <1979> introduced path contraction as a method of saving storage. Path contraction took place if a path in a TAS did not have any branches on it or any propositions classified under any of the nodes except possibly the last node: in this case, all nodes but the first node on the path were deleted. For example, using the same propositions and TH as in figure 3.2, the TAS constructed by this method appears as in figure 3.3.

This method eliminates the overhead of storing straight line paths in TASs. However, it does introduce some problems when accessing a particular node or a sub-hierarchy rooted there if that node has been removed from the TAS under consideration.

The method of finding a node in this form of a TAS is much the same as Goebel's <1977> except that some of the nodes in the path found in the TH are skipped during the descent in a TAS because the corresponding nodes in the TAS have been deleted due to path contraction. The removal of a TAS node causes problems only if it was a leaf node or if one is accessing the sub-hierarchy attached to it.

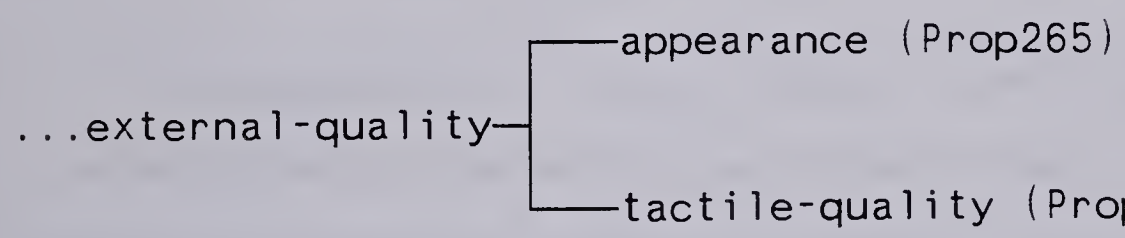


Figure 3.3 Path-Contracted Topic Access Skeleton Fragment

In the case of a leaf node being deleted the propositions that were classified under it are now attached to an ancestor of it. This means that upon reaching a leaf node in a TAS that is not a leaf node of the TH one proposition has to be reclassified to discover under which node the attached propositions are classified.

To access a sub-hierarchy attached to a node, A, in a

particular TAS where A has been removed from that TAS, A's closest descendant in that TAS must be found. A's closest descendant has the instantiated part of A's sub-hierarchy attached to it. This is ensured by the definition of path contraction unless the leaf node case occurs, as described above. Finding A's closest descendant is done by making an ascent in the TH from all descendants of the node in the TAS at which the path to A terminated. If the ascent encounters A then the node that the ascent was made from is the closest descendant of A in the TAS. Determining if the ascent does not encounter A is possible since the ascent must reach some node in the path from the root topic to topic A. Ascending to the root node is easiest since it saves a scan of the path for each node encountered.

The necessity of reclassifying a leaf node proposition is eliminated by the method described here, an adaption of the path contraction method by Schubert et al. <1979>. This method also increases the speed of accessing a topic or the sub-hierarchy rooted there. In the following discussion the TH is restricted to a tree. The discussion can be extended to acyclic digraphs, but the resultant details would obscure the main points.

The first change is the addition of two numbers to the nodes in the TH. The first number, an "identification number" (IN), is the number assigned to the node in a pre-order numbering of the tree. The second number, the

"highest descendant" (HD), is the largest identification number of the descendants of that node. The identification numbers of a node's descendants lie in the interval (IN,HD], the node's "descendent bracket". Figure 3.4 shows the TH fragment of figure 3.1 with descendent brackets attached to each topic. The TAS nodes do not contain descendent bracket numbers, but a link to the TH node they represent. A descendent bracket is accessed from a TAS node by following its link to the TH node it represents and selecting the appropriate fields from the TH node. This change results in

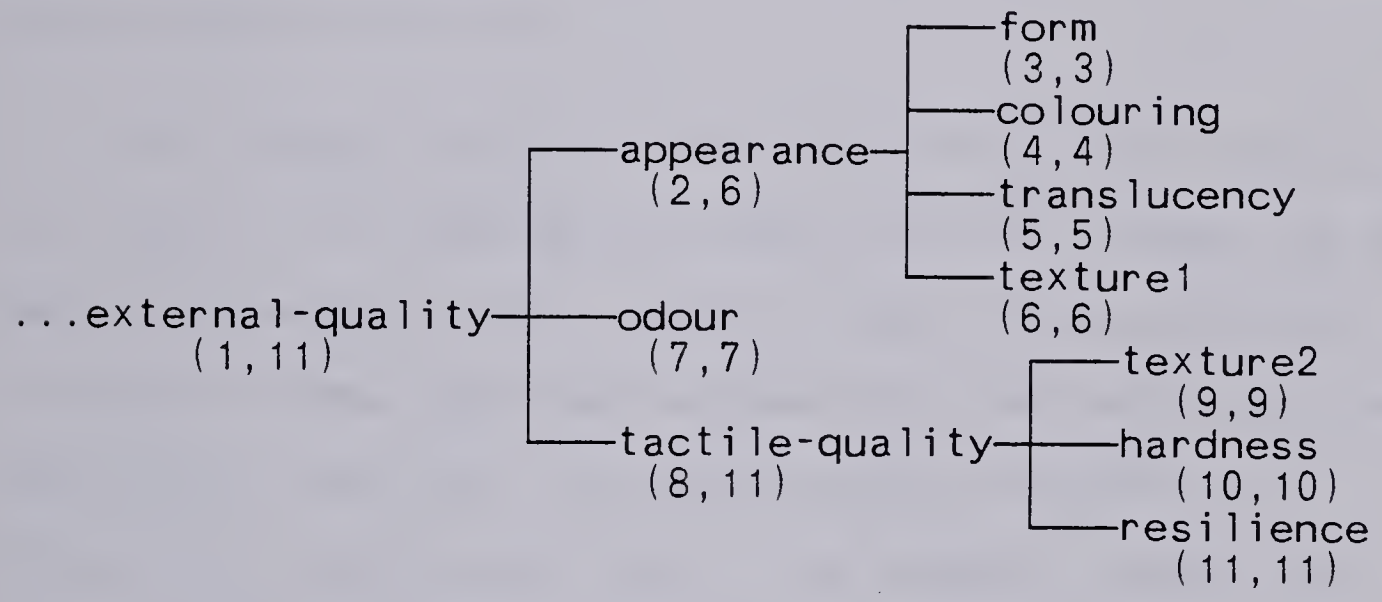


Figure 3.4 Topic Hierarchy Fragment with Descendent Brackets

much easier access to a topic node. No longer is it necessary to ascend in the TH and descend in a TAS to find a TAS node. Finding a TAS node requires only looking up the identification number of the node in the TH and then descending in the TAS under consideration following the nodes whose descendent brackets contain the identification

number of the node wanted until that node is reached. For example, the 'hardness' node is found in a TAS (assume it is a TAS with all nodes of the TH in figure 3.4 present), by looking up its identification number, 10, and then descending in the TAS. The descent follows the path 'external-quality'-'tactile-quality'-'hardness', the nodes that have 10 included in their descendent brackets. Finding an identification number requires a dictionary look up to obtain the topic node and then a field selection on that topic node. Neither are particularly expensive operations and will always require a fixed amount of time independent of the size of the TH.

The second change is the retention of the lowest node of a path to be contracted rather than the highest as was done by Schubert et al. <1979>. Thus all propositions are attached to the TAS node representing the topic under which they were classified, so that reclassification of a proposition need never occur. For example, using the propositions and TH from figure 3.2, the TAS constructed by this method appears as in figure 3.5. As can be seen, straight line paths of TAS nodes terminating in a node that has a proposition attached to it have been contracted to that node. Straight line paths with no propositions on them have been removed.

Several other advantages result from this scheme. Finding a sub-hierarchy of a node, A, in a particular TAS

where A has been deleted from that TAS, is easy with the identification numbering method. This requires finding the closest descendant of A present in the TAS. The descendants

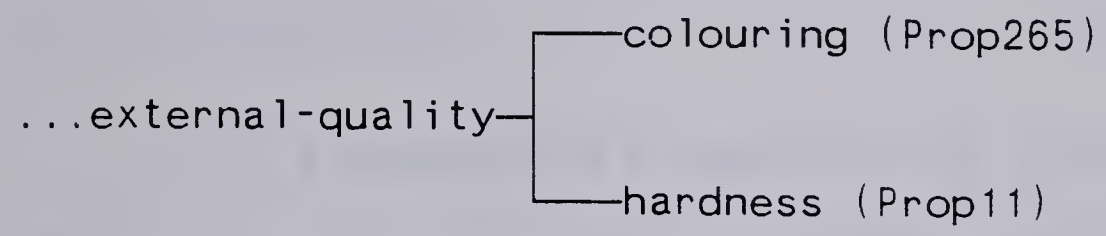


Figure 3.5 New Path-Contracted Topic Access Skeleton Fragment

of the last ancestor of A in the TAS are examined. If one has an identification number that lies in A's descendent bracket then that node is A's closest descendant. No ascents in the TH are required. For example, to find the sub-hierarchy that would be attached to the 'appearance' node in figure 3.5, the descendants of the last ancestor of 'appearance', 'external-quality', are examined. If one has an identification number in the range of the 'appearance' topic node's descendent bracket, (2,6], then it is the sub-hierarchy sought. In this case it is the 'colouring' node. The scheme is also advantageous with respect to maintaining a path-contracted TAS. When inserting a new node into a path-contracted TAS it is necessary to know the ancestor/ descendent relationships between the nodes in the path-contracted TAS and the node being inserted. These relationships are easily obtained without traversing the TH by examining the descendent brackets of the nodes involved.

A description of the algorithm to insert a new node into a path-contracted TAS for a tree hierarchy is presented in appendix A.

3.2.2 Timing Costs

In this subsection a complete TAS[1] method is compared against the list of backlinks method, and then the various TAS methods are compared. In the following discussion let:

n - be the number of propositions classified under a concept.

m - be the number of nodes in the TH.

b - be the average branching factor in the TH.

dt - be the depth of the TH (appoximately log(m)/log(b) for a balanced hierarchy).

da - be the depth of a path-contracted TAS (approximately log(n)/log(b) for a balanced access skeleton). This has an upper bound of dt.

Using a list of backlinks takes n/2 proposition matches on the average to find a particular proposition. The accessing of a proposition using a TAS requires that only the propositions under one TAS node be examined. The TH is designed to partition propositions more or less uniformly across the leaf nodes of the TH, but not exclusively, as

---

[1] All branches and nodes of the TH are present in a complete TAS.

some propositions are classified under internal nodes of the TH as well. A reasonable assumption is that the number of propositions classified under internal nodes is small compared to the total number of propositions classified, so the number of propositions to search is reduced by a factor appoximately of b/((b-1)m). For Schubert et al.'s <1979> sample TH of 53 nodes this factor is approximately 1/40, so the average number of proposition matches to find a particular proposition would be n/80.

Besides this time saving over a list of backlinks, a TAS provides the capability of retrieving topically related propositions at no extra cost, whereas to accomplish this with a list of backlinks requires considerable reasoning and searching. This capability is very useful in answering such questions as "What colour is a lion?". To answer this question any colouring proposition about lions is appropriate.

Timing costs for the various TAS schemes are as follows. For Goebel's original proposal an ascent in the TH, O(dt), is followed by a descent of a full branch of a TAS, O(dt), so total time required is O(dt). Schubert et al.'s <1979> path-contraction method requires an ascent in the TH, O(dt), a descent in a TAS, O(dt)[1], plus possibly one

---

[1] This is still O(dt) even though the TAS is only depth da because the path from the root to A found in the TH must be scanned and it is of length dt.

reclassification, which should be much less than O(dt). Thus total time will be O(dt). The descendent bracket numbering method requires only a descent in a TAS for a total time of O(da). This will be a significant saving in the order of the time whenever da is significantly less that dt (i.e. the TAS descended is sparse compared to the TH); moreover, the need for only one traversal of a TAS and no traversals of the TH under any circumstances reduces the absolute time by at least a factor of two. In the extraction of a sub-hierarchy this factor increases even more with respect to Schubert et al.'s <1979> method if the root of the sub-hierarchy has been deleted from the TAS under consideration. This occurs because Schubert et al.'s <1979> method requires extra TH traversals to ascertain the ancestor relationships necessary to find the closest descendant of the deleted node.

There is one hidden time cost in the descendent bracket scheme. It is the time required to renumber the TH on addition of a new topic. This will be discussed in greater detail in section 3.4.

3.2.3 Storage Costs

The list of backlinks method is the cheapest in storage, O(n). Schubert et al. <1979> have shown that path contraction will only introduce a factor of two overhead over the list of backlinks method. This is an acceptable

overhead considering the reduction in time for proposition access and for the topic structuring of the propositions.

Storage costs for the descendent bracket numbering will be 2m over storage costs for a non-numbered hierarchy since TAS nodes do not contain the bracket numbers, but a link to the topic node they represent.

3.3 Concept Hierarchy

The concept hierarchy (CH) enables associative accessing of concepts by their properties. A property of a concept is indicated by a monadic type predication about that concept. The CH is built in a manner analogous to the TH, except that instead of sub/super topic predications, sub/super concept predications are used to build the hierarchy. As with the TH, the CH merely defines the structure. To actually use the structure, a concept access skeleton (CAS) is used. CASs use the same data structure and thus the same building algorithm as TASs.

To explain "associative access" and to partially motivate the CH and CAS a single net is considered first. The final motivation comes from examining the role CASs play in subnets.

To access an instance (individual) associatively means to access that individual by specifying a property of it. For example, to find something that is an elephant by being

able to access from 'elephant' all things that are elephants. This sort of question can be answered by scanning all individuals, checking to see if they are an elephant. This becomes prohibitive when the number of individuals becomes large. A CH and its attendent CAS allow rapid associative access to individuals. For example, to access an elephant, the elephant node in the CAS is accessed[1]. This node has a list of all things that are known to be elephants attached to it[2]. In this particular case it can be argued that "instance" backlinks from the predicate 'elephant' to all instances of it would suffice, but if the question is extended to "What animals do you know?" the "instance" backlinks will not do because very few, if any, instances of animals will be directly predicated to be animals. Rather they will have been predicated to be dogs, cats, etc. and the generic dog or cat will have been predicted as being a sub-concept of 'animal' (probably not directly but through a few layers such as, 'dog' to 'mammal-quadruped' to 'mammal' to 'higher-animal' to 'animal'). The query "What animals do you know?" can be answered using an exhaustive search of all sub-concepts of 'animal', checking each for instances associated with them.

---

[1] This is done in the same fashion as a TAS access so it is fast.

[2] In actual fact, attached to the elephant node is a list of all the propositions which state that an individual is an elephant. The individual can be quickly extracted from the proposition.

This is very time consuming. A CAS access is faster for several reasons. It contains only those nodes that have instances attached to them plus a few internal nodes to maintain the structure (see TASs), thus searches down branches with no individuals attached are avoided. Also because a CAS is a path-contracted access skeleton a minimum number of nodes are traversed, whereas a CH explicitly represented in the knowledge base requires that all nodes be traversed.

Turning now to subnets, it should first be pointed out that individuals in a subnet should be accessed only by a CAS unique to that subnet and not by the main net CAS. If individuals in a subnet were accessed by the main net CAS, all individuals would be mixed together. A check would always be required to make sure that the individual found was in the net currently being used.

The final motivation for the CH/CAS structure is that in subnets a complete CH is often not explicitly stated. For example, in the story of "Little Red Riding Hood", when the wolf enters the scene, the facts that he is a mammal and that mammals are animals etc. are not mentioned. It is assumed that the listener will make these assumptions from his "real world" knowledge. Anything that is "extraordinary" about the wolf, such as his ability to talk, is made clear. After listening to the story most listeners, if asked "What animals are in the story?" would reply "a

wolf". It is possible to obtain this answer from the knowledge base without using a CAS, but at considerable expense. In this case the search would proceed as follows. A search for the virtual 'animal' concept in the subnet of the story of "Little Red Riding Hood" is done. When this fails, the sub-concepts of 'animal' in the main net are found and then each of these in turn is sought in the story subnet. The search for these concepts follows the same procedure as the search for the 'animal' concept. If any of these concepts are found in the subnet then a check would be made to see if any individuals of that concept existed in the subnet. When all of the sub-concepts of 'animal' in the main net had been searched for in the subnet and 'animal' sub-concepts in the subnet have had their sub-concepts searched, then the answer "a wolf" could be output. This procedure is obviously time consuming. In this particular case, a CAS would probably have the sub-hierarchy for 'animal' only two or three levels down. Thus, accessing it would be very fast. The sub-hierarchy would probably only be the 'wolf' node due to path contraction (and assuming that humans are put into a special catagory by themselves, which is likely considering the importance generally attached to them).

It must be noted at this point that a CAS really only returns candidates for the particular property sought. This occurs because all CASs are based on a single CH, the CH of the main net. This CH may not correspond exactly to an

explicit one in a subnet or for that matter to the explicit one in the main net. For example, the CH may have 'whale' as a sub-concept of 'mammal' and John may believe that whales are fish. If the question "What mammals does John know?" is asked then any whales John knows, say Spot, are returned along with other mammals from the CAS search. To completely answer the query each individual must be checked to see if it has the correct property. This is a straightforward part of the property and relationship inheritance problem which is discussed in chapter 5.

3.4 Combined Hierarchy

Since the TH and CH are built on the same structure there is no reason not to make them into one structure by joining them at their roots. One half of this super-hierarchy organizes knowledge by its relevance to a particular concept (the TH); the other half organizes concepts by the properties assigned to them (the CH). One reason to do this is that the CH can be used to more finely classify "specializations" of concepts (see chapter 4).

One problem becomes evident when the CH is structured in the same way as the TH. It is caused by the fact that the CH will typically grow fairly quickly as new concepts are added to the system. Each addition will require a renumbering of the nodes in the combined hierarchy. This is an O(n) operation where n is the number of nodes in the

hierarchy. In a large system this becomes time consuming. A solution to this is to leave gaps in the numbering scheme so that some new nodes can be added to the super-hierarchy without requiring a renumbering of the complete hierarchy. This does not reduce the time spent to do one renumbering, but it does reduce the total number of renumberings required. An algorithm for a fairly simple scheme of this nature is presented in appendix F.

Chapter 4

CLASSIFICATION

The TH and CH have been shown to be useful structures for accessing knowledge once it has been attached to them. This chapter describes the scheme which classifies a proposition-concept pair with respect to these structures. Since two structures are used there are two forms of classification. One assigns a topic or topics as the classification of a proposition-concept pair. It is based on the topical classification schemes proposed by Goebel <1977> and Schubert et al. <1979>. Some extensions and alterations are made to these two schemes to accommodate the CH/CAS structure and to correct some errors in these methods. The other form assigns a concept in the CH and a net as the classification of a proposition-concept pair. This is the form which organizes concepts to allow associative access by the properties known about them. The first section deals with the classification of non-modal propositions[1]. The second section deals with modal predicates and the problems they introduce with respect to classification.

---

[1] A non-modal proposition is a proposition that does not contain a modal predicate or operator.

4.1 Non-modal Classification

The classification scheme for the individual clauses created by normalization basically follows the rules laid out in Schubert et al. <1979>. A few changes are required in order to accommodate the CH/CAS structure. These are described in section 4.1.2 and 4.1.4.

Classification of a proposition is carried out with respect to every existential concept in the proposition. The proposition is not classified with respect to universally quantified concepts since the proposition does not say anything about them, but rather says something about the predicates applied to them. For example, the proposition

∀x[[x Kangaroo]=>[[x marsupial]&[x herbivore]]]

says something about Kangaroo things, marsupial things and herbivore things not about all things. Sometimes the classification process does not yield a topic for a proposition-concept pair. This occurs when the backlink from that concept to the proposition was considered unimportant. For example, the backlink from 'elephant' to

¬[Clyde elephant]

is not useful. Obviously, if future research shows that the backlinks lost are important, the appropriate changes to the classification scheme can be made to include them.

This section is organized into several subsections. Each deals with a particular form of proposition-concept

pair, describing how each form is recognized and the type of the classification. The first three subsections deal with classification in the TAS and the last with classification in the CAS.

4.1.1 Generalization

This topic classifies propositions that give the type of a concept. Type cannot be distinguished by syntatic form alone (see chapter 5). A check for a type predicate must be made. This is straightforward since the indicator link from a type predicate points to the 'generalization' topic. There are two syntatic forms a generalization proposition can take. One gives the generalization of an individual, the other a generalization of a generic concept. For example,

[Clyde elephant]

∀x[[x elephant]=>[x mammal]]

give the type of 'Clyde' and 'elephant' respectively. The general rules are <Goebel 1977, Schubert et al. 1979>:

1) If an existential concept occurs as argument of a type predicate then the proposition is classified with respect to the existential concept as a generalization proposition.

2) If a universal concept occurs as argument of a

negatively occurring[1] type predicate and of a positively occurring type predicate then the proposition is a generalization predication with respect to the negatively occurring type predicate.

4.1.2 Specialization

This topic contains propositions that give a special case (subset), that is not an instance, of a concept. The scheme proposed by Schubert et al. <1979> included instances under this topic, but now instances are referenced under the CH/CAS structure. For example,

∀x[[x elephant]=>[x grey]]

says that elephants are a special case (subset) of grey things. Another example is

∀x[[x elephant]=>[x mammal]].

This proposition says elephants are a special case of mammals. The general rule <Schubert el al. 1979> is that if a universal concept occurs as an argument of a negatively occurring type predicate and of a positively occurring predicate then the proposition is a specialization with respect to the positively occurring predicate. Under the 'specialization' topic the proposition is further classified

---

[1] "Negatively occurring" predicate means a predicate that occurs in a negated atom when the proposition is put into CNF.

using the CH structure. The proposition is classified by the type predicate. In the combined hierarchy the CH is hung from the specialization topic node. The first example would be classified under the 'elephant' TAS node of the 'grey' concept, similarly the second example would be classified under the 'elephant' TAS node of the 'mammal' concept.

4.1.3 Topics

This branch of the hierarchy organizes attributes and relationships of concepts. These are represented by non-type predicates. Some examples of attributes are:

∀x[[x elephant]=>[x grey]]

which is classified under the 'colouring' topic with respect to 'elephant' and

[Clyde grey]

which is classified under the 'colouring' topic with respect to 'Clyde'. The general rules for attributes are <Goebel 1977, Schubert et al. 1979>:

1) If a universal concept occurs as an argument of a negatively occurring type predicate and of a positively occurring non-type predicate then the proposition is classified with respect to the negatively occurring type predicate according to the indicator link(s) attached to the positively occurring non-type predicate.

2) If an existential concept occurs as an argument of a

non-type predicate then with respect to that concept the proposition is classified according to the indicator link(s) attached to the predicate.

Relationships generally have indicator links that depend on the argument position (section 3.1.1). The same rules as above are applied to relationships, but with the added constraint that the argument position of the concept involved must match the argument postion on the indicator link.

4.1.4 Instances

An instance of a predicate is a constant which appears as the argument of a predicate. For example, in

[Clyde elephant]

∃y[y elephant]

[Clyde grey]

'Clyde' is an instance of 'elephant' and 'grey', and 'y' is an instance of 'elephant'. An instance predication is classified in a CAS under the node associated with the predicate. If there is no CH node associated with the predicate then the predication is classified in the predicate's TAS under the 'specialization' topic. If a CH classification is produced, the CAS used depends on the net in which the proposition is asserted. This classification corrects an error in Goebel's <1977> classfication scheme, namely that backlinks from a concept to an instance of it

were lost, thereby making questions such as "What is an example of an elephant?" hard to answer.

4.2 Modal Predicates

There are two classes of modal predicates or predicates that have propositions as arguments. They are ones that form conceptions of the world (e.g., hope and belief) and ones that do not (e.g., causes, credibility). The sub-propositions embedded in the latter predicates are classified in the main net as if they were top level propositions. The fact that they are embedded in another proposition is indicated by a backlink from them to their embedder. The predicates that form conceptions of the world are of more import to this thesis, thus the rest of this section is devoted to describing how they are handled.

A modal proposition with an individual first argument and propositional second argument is classified as follows. First the proposition is classified with respect to the first argument by the indicator link attached to the modal predicate. Then the enclosed propositions are classified using the same method described for top level propositions, but in the subnet of the first argument of the modal predicate. The classification is not attached to main net concepts but to the virtual concepts in the subnet or to the CAS associated with that subnet. As can been seen, this is a recursive definition and as such can handle such sentences

as "Mary hopes that John believes that she likes him.".

There is one decision to make about modal classification. It is choosing what part of the proposition is referenced by the classification. There are two parts of the proposition that could be pointed to by the classification, namely:

1) The enclosed proposition.

2) The top level (modal) proposition.

If the classification points to the enclosed proposition then it may be difficult to find the superordinate proposition. This occurs because the enclosed proposition may be a sub-proposition in many propositions, thus it will have backlinks to many enclosing propositions. Finding the one particular to the subnet of interest requires a linear scan of the backlinks. If, on the other hand, the classification references the top level proposition then this problem vanishes, but then the problem of finding the sub-proposition that created the classification appears. The solution to this is straightforward. The sub-proposition is found by traversing the same number of modal predicates as subnets traversed to arrive in the subnet where the classification was found. For example, the proposition

[John believes [Mary hopes [John likes Mary] .9] .92]

is classified under the virtual concept of Mary, as say an 'emotional-object' proposition, in the subnet which was

arrived at by going from the main net to John's net to the net attached to the virtual concept of Mary in John's net. The sub-proposition which created the classification is the one found by traversing two modal predicates (number of subnets entered), 'believes' and 'hopes' in this case. The sub-proposition found is [John likes Mary]. Because of the above reason the second reference point for classification of modal predications is used.

Chapter 5

PROPERTY AND RELATIONSHIP INHERITANCE

This is the problem McDermott <1975a,b> named the "Symbol-Mapping" problem. The "Symbol-Mapping" problem appears when knowledge is stored at the highest point possible in a generalization hierarchy. For example, the knowledge that a cat has four legs that are joined to its torso at certain points is not associated with the 'cat' concept, but is associated with a more general concept, such as 'quadruped'. Thus to find out where a cat's legs are attached, the legs of some generalization of cat have to be accessed and their attachment points and relationships propagated (inherited) down to the cat concept. McDermott <1975a,b> was mainly concerned with propagating knowledge down a generalization hierarchy from generic concept to generic concept or from generic concept to individual concept (e.g., from mammal to elephant to Clyde the elephant). Another part of this problem is propagating knowledge from dependent superordinate to corresponding dependent subordinate concepts (e.g., from the quadruped's torso and legs to the cat's torso and legs; or from the generic mammal's head to the generic elephant's head to Clyde's head). This is the problem considered in this chapter. In the literature this problem has been approached as the problem of inheriting properties and relationships from dependent parts of a concept to the corresponding

dependent parts of a subordinate concept <Raphael 1968, Hayes 1977, Schubert et al. 1979, Schubert 1979>.

The problem can be broken down into two sub-problems, matching the corresponding dependent concepts of a concept and its generalization, and accessing the dependent concepts of a concept. Matching corresponding dependent concepts of a concept and its generalization (e.g., Clyde's right front leg to the elephant's right front leg; or the elephant's trunk to the mammal's nose) is outside the scope of this thesis since it requires fairly complex reasoning processes. As can be seen from the above examples, information other than the type information is often required (e.g., right front leg); or the type of one of the dependent concepts may be a generalization of the type of the other (e.g., trunk to nose). What is examined in this thesis is a structure to facilitate rematching two dependent concepts once the reasoning matcher has decided, possibly tentatively, that two dependent concepts correspond. The next time this relationship is needed for reasoning it is quickly available via this structure with no reasoning required.

This chapter is divided into two sections. The first defines what is meant by a "generalization hierarchy". The second looks at four proposed solutions to this problem. The first two are from the literature. The third solution failed, but provided some of the motivation for the last solution.

5.1 Generalization Hierarchy

A generalization hierarchy is a type hierarchy going from most general type at the root of the hierarchy to most specific type at the leaves of the hierarchy. The generalization hierarchy is built by predications giving the superordinate type of a type predicate. For example,

∀x[[x elephant]=>[x mammal]]

sets 'mammal' as a direct generalization of 'elephant'. A primitive generalization hierarchy fragment is shown in figure 5.1.

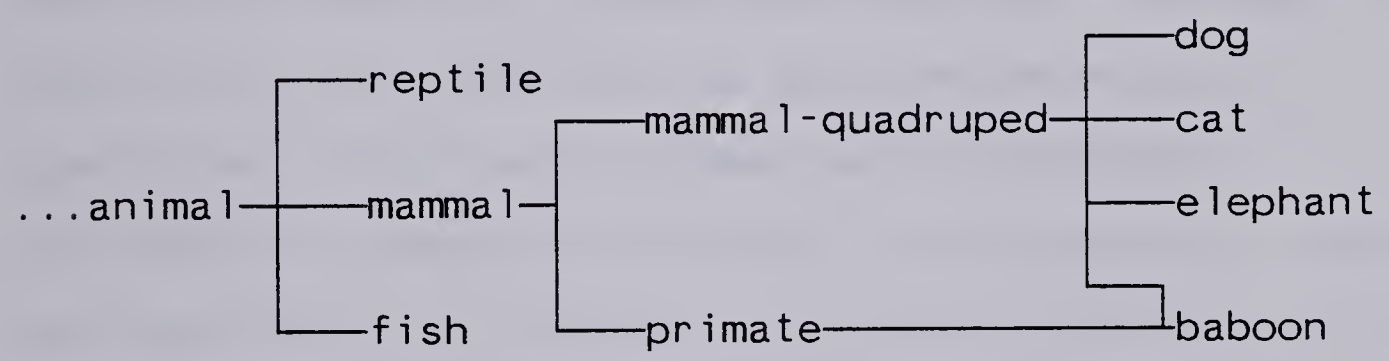


Figure 5.1 A Primitive Generalization Hierarchy Fragment

Generalization hierarchies are not necessarily trees. For example, consider where baboon fits into the generalization hierarchy in figure 5.1. It is a primate and a mammal-quadruped. The generalization hierarchy is represented explicitly as propositions in the knowledge base, so there are no problems representing tangled type hierarchies.

5.2 Solutions

This section discusses several schemes for matching and accessing parts. Two methods from the literature are described and the problems with each are discussed. The third method, the "subnet" scheme, a mixture of the above two, is introduced to overcome some of the problems. The "subnet" scheme still has some major failings, but it provides the germ for the last scheme.

Schubert et al. <1979> borrowed the variable sharing part of Hayes' <1977> depiction scheme and put it into the more formal setting of predicate calculus. The basic idea behind variable sharing is to have one universally quantified concept and a common set of dependent existentially quantified concepts to represent all concepts and their parts in a concept hierarchy, such as the higher animal hierarchy. The particular animal and parts represented would depend on the "viewpoint" (i.e. what animal is being talked about). Using Schubert et al.'s <1979> example, figure 5.2, plus the appropriate super-concept predications (e.g., emus are birds), 'y' is an emu head when viewed from the emu viewpoint, but changes to an owl's head when looked at from the owl viewpoint. Because the variables are shared across viewpoints, the matching of parts from concept to super-concept is already done. Thus, it seems that relationships and properties are easily transferrable. For example, it seems that the fact that an

emu's or an owl's head is attached to its neck is readily available from the bird viewpoint. This is the case if the properties and parts can be found, but this is difficult

```
∀x]y]z[[[x bird]=>[[y part-of x]&[z part-of x]&[y head]
&[z neck]&[y joined-to z]]] &

[[x emu]=>[[y head1]&[z neck1]&[z small]&[z long]]] &

[[x owl]=>[[y head2]&[z neck2]&[y big]&[z short]]]]
```

Figure 5.2 Variable Sharing

because backlinks from the shared variables to the propositions using them lead to all propositions about all types of entities sharing those variables. Thus, the knowledge about the head and the neck being joined is most efficiently accessed via either the 'bird' concept or the 'joined-to' concept and not from the head, 'y', or neck, 'z' concept as would be supposed. The fact that an entity's parts, sub-parts, etc. and knowledge about them must be directly accessible from the entity leads to unstructured access to the parts and parts knowledge of an entity. Yet, when one is reasoning about an elephant's toes one does not want the facts about elephant tails to be as prominent as the elephant toe facts. One last problem is that variable sharing does not take into account individuals since an individual cannot be denoted by a universal node.

Hayes' <1977> uses depictions to represent a generic concept, its dependent concepts and the links between them.

A depiction contains a generic node for the generic concept, existential dependent nodes for the dependent parts, and relationships between the nodes. For example, the depiction for a human would contain a generic node for 'humans', existential dependent nodes for human arms, legs, etc., and generic relationships between the nodes, such as the arms are part of the human and the arms are connected to the torso. Nodes are shared between depictions in a concept hierarchy. For example, the human torso node would be the same node as the animal torso node and the dog torso node. This node-sharing is logically equivalent to the variable-sharing method of Schubert et al. <1979>. The problem of unstructured access to sub-parts of generic concepts is solved in Hayes' system by having only the major parts in a top level depiction. Each of these major parts will be the generic node for another depiction which will list its major parts and so on to any level of detail. For example, the dependent node in the human depiction for human arm would be a generic node in the human arm depiction. The human arm depiction would have dependent nodes representing parts such as fingers, hands and wrists. This only slightly alleviates the problem of accessing knowledge since it is still not accessible from the nodes in a depiction, but only from the depiction itself. Hayes' <1977> solves the problem of individuals inheriting relationships from generic concepts by providing binders. A binder lists all the parts which make up an individual and binds them to their corresponding

generic parts. Since this does not structure the parts of an individual, the same accessing problem exists as with Schubert et al.'s <1979> scheme.

The main problem common to these two representations is that relationships between generic nodes, such as mute swans are larger than whistling swans, cannot be stated because the same concept is used to represent different entities in the same hierarchy.

The "subnet" scheme solves this problem by retaining a distinct universally quantified concept for each predicate. A subnet is generated for each universally quantified concept that occurs as an argument of a negatively occurring type predicate. What this means is that all knowledge about a particular concept is put into that concept's subnet. This allows sharing of variables via their names. For example, the "toe" node in the elephant subnet would have the same name as the "toe" node in the creature subnet, assuming that elephant is a sub-concept of creature. This provides for rapid matching of corresponding parts from super-concept to sub-concept. For example, to find more about elephant toes, one looks in the creature subnet under the name that the elephant toe has in the elephant subnet and all the knowledge about creature toes is available. Since the variables are shared by name only, backlinks from dependent concepts reference only those propositions relevant to that concept. Thus, knowledge about a part is

easily accessible from it and does not have to be accessed from the entity.

It may seem that subnets would require a lot of storage. Admittedly, there is some overhead incurred by using subnets, but this is at least partially offset by the removal of part of the antecedent for each implication. The part removed is the negative type predication which defines the subnet. It can be removed because it is implied for any propositions occurring in the subnet.

There is a major problem with the subnet idea discussed thus far. Subnets do not take individuals into consideration, because subnets are generated only by universally quantified concepts. Thus an individual does not have a subnet of its own. Consequently propositions about an individual exist in the main net, rendering impossible the sharing of variables by name between an individual and its generic super-concepts.

Sharing a dependent concept name between two subnets can be more formally thought of as using the same Skolem function name for the two concepts <Schubert 1978>. This prompts the last scheme, which is to use Skolem function tables to link corresponding dependent concepts <Schubert 1979>. In the "subnet" scheme corresponding dependent concepts were linked by their name; in this scheme corresponding dependent concepts are linked by their function name. The fashion in which it is envisioned to

work is as follows. For example, assume the statement "All animals have a head." generates the Skolem function 'f' for the existentially quantified head of the generic animal. When the statement "All thrushes have heads." is entered the Skolem function is also called 'f'. How these two functions receive the same name is discussed later. It must be mentioned at this point that 'f' does not correspond to the 'head-of' concept. The 'head-of' relation is not necessarily single valued, whereas 'f' being a function is single valued. This is necessary so that such entities as two headed turtles can be talked about. One head would be accessible by the function 'f' and the other by some other function. With this arrangement for dependent concepts of all entities, corresponding dependent concepts between different levels in the generalization hierarchy can be matched quickly by merely matching function names. Once corresponding dependent concepts are found the relationships and properties are easily inherited. As a bonus, this method works as efficiently between any two concepts of the generalization hierarchy as between adjacent levels. However, there are a few problems that need to be examined.

How is the initial unification of function names accomplished, such as the correspondence between the Skolem functions for thrush head and bird head? This falls under the problem of matching corresponding dependent concepts. As was mentioned before, this is outside the scope of this thesis, but a brief discussion of how this might be

accomplished is given below. The outline follows Schubert's <1979> proposed method. A form of description matching will have to be used. For example, if the sentence "Clyde's right front leg was cut." is input, a referent for "right front leg" is sought. The search will start at Clyde and then expand vertically in the generalization hierarchy from Clyde and possibly horizontally in the generalization hierarchy as well. If a referent is found then the function name it uses is tentatively used for Clyde's right front leg as well. If a referent is not found then a new function name is generated. Note that for concepts which are not a leaf of the generalization hierarchy the vertical search for a referent must be made below and as well as above the concept.

Are individuals included in this scheme? The fashion in which dependent concept relationships are usually written for individuals does not make self evident the integration of individuals into the Skolem function table scheme. For example, the statement that Nibor, the robin, has a head is usually represented as follows.

∃x[[x head]&[x part-of Nibor]]

In this representation 'x' is a Skolem function of zero arguments (i.e. a constant) and cannot be linked via function name, since it does not have one, to the generic robin's head. If the above statement is rewritten as

[[(t Nibor) head]&[(t Nibor) part-of Nibor]]

then since Nibor's head is now a function of Nibor it can be

linked to the generic robin's head via its function name. This requires an extension to Skolem function tables as presented in section 2.3. They were attached only to universally quantified concepts. The extension is to allow them to be attached to existentially quantified concepts. With this extension a function table can be attached to Nibor and Nibor's head can be indexed in this table under a function name. The reasoning dependent concept matcher can now be applied to Nibor's head and the function name for the robin's head can be used to index Nibor's head in Nibor's function table. It is not being suggested that dependent constants actually be entered this way, but that they be translated to this style by some internal mechanism. At the moment this along with matching corresponding dependent concepts is done by hand.

How are a particular function name and its corresponding value found from a concept? This is straightforward and efficient if a hash table is attached to each concept. The table is indexed by function name. The value in the table corresponding to a function name is a pointer to the existential variable that represents that function for the concept to which the table is attached.

Chapter 6

IMPLEMENTATION AND UTILIZATION

6.1 Data Structures

This section describes how the extended semantic network graphic representation of Schubert <1975> and the accessing and organizing structures imposed on top of the semantic network are translated into computer data structures. Since these structures are of necessity intermingled, they cannot be described in isolation from each other so some forward references are unavoidable. The data structures that are used to represent propositional knowledge are discussed first, followed by the organizational structures and some utility structures.

6.1.1 Knowledge Representation Structures

There is a one-to-one correspondence between nodes in the graphic form and records in the computer data structure form. Arcs in the graphic form are represented by pointers in the computer. The graphic form explicitly labels arcs either with letters or by the type of line used to draw the arc. This is not necessary in the data structure form as the type of the arc is implicitly given by the origin of the corresponding pointer in the data structure.

There are two basic types of records in the data structure, concept and proposition records. This contrasts with Goebel's <1977> representation where separate types of records are used for predicates (constants only) such as 'grey' and 'elephant', and for non-predicates such as 'Clyde' and variables. This lack of uniformity of representation of concepts results in two difficulties. First, predicate variables are not allowed. Second, the system has to decide whether an argument of a predicate can itself be a predicate, in which case it should be put in with the predicates, or if the argument is an instance which should be put in with the non-predicates. For example, if one has no a priori knowledge of colours or of the predicate RED the proposition

[RED COLOUR]

does not tell one that RED can be a predicate. These two difficulties are overcome by having only one type of concept record.

Concept records have several fields. Those that support the knowledge representation are as follows:

1) quantification.
2) name - the name of the concept. In the computer representation all concepts are given names. Variables are given a unique "internal" name.
3) scope inclusion - this has several functions depending on the concept. If it is a universally quantified concept or an existentially quantified

concept that has dependent concepts (e.g., Nibor) then this field points to a Skolem function table which indexes by function name the concepts dependent on this node. If it is an existentially quantified concept that depends on some concept then this field contains the name of the function of which this node is a value and a list of the argument(s) of the function which produce this node. If the concept is dependent on a subnet then this field contains a pointer to that net.

The rest of the fields are attachment sites for the various organizational structures. They are:

4) for existential concepts, a pointer to the concept's topic access skeleton. For universal concepts, a pointer to the type predicate with which it is associated.
5) a list of topic indicators. Each list element includes the argument number to which the topic applies and the relevance of the predicate to the topic.
6) a pointer to a hierarchy node if the concept is part of either the topic or concept hierarchy.
7) a list of universally quantified concepts associated with this concept

In section 2.4 it was argued that to provide access to knowledge inside different conceptions of the world a virtual concept, a hook for access structures, was needed.

It has the following fields.

1) quantification.
2) name - pointer into the local dictionary.
3) scope inclusion. This field never points to a net since it is the main concept which contains the net inclusion information. Otherwise, it has the same function as the scope inclusion field of a main concept.
4) a pointer to the node's topic access skeleton.
5) pointer to the main concept of which this is a virtual concept.

Since all accessing is via the concepts, proposition records have only the fields necessary to specify their structure. The fields are:

1) name.
2) scope inclusion.
3) predicate.
4) a list of arguments in the order corresponding to the order in the input expression.
5) an ordered list of time arguments.
6) assertion time. The value of the internal clock when the proposition was asserted.
7) a list of back pointers to super-propositions.

Logical operators are represented by a set of reserved numbers. When a proposition is formed using a logical operator the number of that operator is used in the

predicate field of that proposition node.

The scope inclusion field is used only by modal operators or predicates that are not propositional attitudes. As mentioned in section 2.4 whenever a propositional attitude predicate is encountered the enclosed propositions are placed in a subnet associated with the first argument of the modal predicate. The subnet dictionary has entries for all concepts used in the subnet. To determine if a concept is dependent on that subnet the scope inclusion field of the main concept is accessed. If it points to the subnet then that concept is dependent on that subnet.

6.1.2 Knowledge Organization Structures

As pointed out in section 2.4 subnets have exactly the same structures as the main net, so the main net and subnets are represented by a record with the following fields.

1) dictionary. It allows "by name" access to concepts used in this net.
2) concept access skeleton. It allows associative access to the concepts in the net.

The super-hierarchy is built using hierarchy records. Each hierarchy record consists of the following fields.

1) name.
2) parent. Since the structure being represented has been restricted to a tree only one parent is ever

present.

3) list of offspring. Each list element includes the relevance of the offspring to the hierarchy node.

4) a pair of numbers defining the descendent bracket. The first number is the node's identification number.

The access skeleton records are as follows.

1) a pointer to the hierarchy record that this record represents.

2) a list of descendants. These may not be immediate descendants because of path contraction.

3) a list of items classified under this node.

6.1.3 Utility Structures

Tables are used for net dictionaries and for function tables. A table has the following fields:

1) size. number of slots allocated for the table.

2) fill. number of filled slots in the table.

3) an array of length "size" of slots. Each slot consists of a name and a pointer to an object with that name.

The structure that remains to be discussed is the sentential form hash table. It references all sentential forms in the knowledge base. It is simply an array of pointers to proposition records. It is used to prevent replication of sentential forms in the knowledge base (see

section 2.5).

This completes the description of the data structures used to store and access knowledge. The next section describes the programs that use them and how they are organized.

6.2 Utilization

This section describes the main components of the system. A description of the action and various sub-modules called by each main component of the system is given. The system is divided into three top level sub-systems. They are the structure builder, the structure user, and miscellaneous utility routines.

6.2.1 Structure Builder

There are two sub-modules to this sub-system. One which builds the internal representation of propositions, and the one which builds and modifies the access structures.

6.2.1.1 Proposition Entry

There are six phases to entering a proposition into the knowledge base. They are described below.

1) Parsing

This consists of parsing the surface text and building

an equivalent internal structure using the analog of the graphic representation described in section 2.1. The grammar used is similar to Goebel's <1977>. A formal description of it is given in appendix B. Variables are represented as temporary concepts since it is possible that they will be replaced by concepts in the knowledge base.

2) Normalization

This phase puts the internal structure created by the parser into modal conjunctive normal form.

3) Variable Replacement

This is done in the following order. Constants (Skolem functions of zero arguments) are replaced by unique permanent concepts. Next temporary universally quantified concepts are replaced. This is accomplished by finding the negative type predication about the temporary universally quantified concept, accessing the permanent universally quantified concept associated with the type predicate, and finally replacing the temporary by the permanent universally quantified concept. The last variable concepts to be replaced are Skolem functions with more than zero arguments. At present they are replaced by uniquely named existential concepts. The Skolemization of an existential concept is not lost but is added to the Skolem function tables of the permanent universally quantified concept(s) upon which it

depends.

4) Proposition Replacement

The knowledge base is searched for the clauses resulting from normalization and variable replacement. The use of the sentential form hash table and method of searching for a clause was discussed in section 2.5. Any clauses that are completely found in the knowledge base are removed from further consideration.

5) Classification

Each clause that is left is classified in turn. The classification routine tries to classify a clause with respect to every existential concept occurring in the clause. The classification of a clause is returned as a list of "binders". Each "binder" binds either a concept or a net to a set of super-hierarchy nodes. The hierarchy nodes form the classification of the clause with respect to the bound concept or net. Another binder is produced which binds each clause to its classification. This is passed to the last stage of the proposition entry, assertion. The classification routine does not add the classifications to the appropriate TAS or CAS as it goes. This is done so that the classification of a proposition can be obtained without altering the knowledge base. Thus, it can be used to find the classification of clauses that are to be searched for in

the knowledge base.

6) Assertion

The last stage of adding a proposition to the knowledge base is asserting it in the knowledge base. This is done by adding the classifications found by the classification stage to the appropriate TASs and CASs. The addition of any new access skeleton nodes is done via the access skeleton building algorthim described in section 3.2.1. Thus path-contracted access skeletons are maintained thoughout the system.

6.2.1.2 Access Structure Entry

This module is split into two sub-modules, hierarhcy building and function table building and modifying. The latter is the hand version of the description matcher mentioned in sections 2.5 and 5.2.

Hierarchy building includes building the super-hierarchy and adding topic indicators from predicates to topic hierarchy nodes. Since both of these structures are represented in a special fashion in the system they are entered with a special grammar. The grammar allows relevance concepts for the hierarchy and argument number and relevance for topic triggers. Appendix C contains a formal description of this grammar.

Function table modification is allowed for universal and existential concepts. The permitted modification is to change a function's name. Function table building lets constants be made explicit functions of other constants (e.g., Nibor's head of Nibor). The grammar to specify these modifications is shown in appendix D.

6.2.2 Structure Using

There are several inputs which use or examine the structures in the knowledge base. Most are of a nature useful for debugging the structure builder, since they print the structures in an explicit fashion[1]. The more interesting one takes the form of a simple query language.

The query language is the input for a query answerer. The answerer shows the use of generalization hierarchies, CASs, function tables and TASs, in subnets as well as in the main net. It is discussed in section 6.2.2.2. The language it accepts is discussed in the next section.

6.2.2.1 Query Language

The query language takes the form of a '?' followed by a "simple proposition" or a "simple proposition" embedded in an arbitrary number of propositional attitude predications.

---

[1] These are described in appendix E

A "simple proposition" is a proposition which has no embedded sentential forms. For example,

[Tom wolf]

is a simple proposition while

∀x[[x wolf]=>[x mammal]]

is not a simple proposition. Thus queries are limited to queries about individuals or concepts dependent on individuals in any net. The query may have "query variables" in any place a concept would normally appear. A "query variable" is denoted by a string of characters whose initial charater is a '?'. A "query variable" will match any concept in the knowledge base. If the query of which it is part matches a top level proposition in the knowledge base then it is "bound" to the concept which it matched in the proposition. Some examples of queries with English translations are shown in figure 6.1.

?[Clyde grey]

Is Clyde grey?

?[?x likes Clyde]

Who (what) likes Clyde?

Figure 6.1 Simple Queries

Predicates are also allowed to be represented by "query variables" with some additional information. A topic must be given with the predicate "query variable". This

specifies the area of the predicate. Examples of this are shown in figure 6.2. If the question is for individuals with a certain property then a CH node must be specified.

```
?[Clyde ?x{tp.colouring}]
What colouring does Clyde have?
(What colour is Clyde?)

?[Clyde-head ?x{tp.physical-relation} Clyde-body]
Name a physical relationship that holds between Clyde's
head and his body.
```

Figure 6.2 Predicate Queries

This is because individuals are accessed via their properties via the CAS of the net upon which they depend for existence. An example of this is shown in figure 6.3. The query in figure 6.3 is in fact equivalent to the query

```
?[?x elephant]
```

since 'elephant' specifies both a topic and a concept hierarchy node, namely the generalization node and the elephant node respectively. The need for the more complex form, as in figure 6.3, is discussed later.

```
?[?x ?y{tp.generalization:elephant}]
What elephants do you know?
```

Figure 6.3 Concept Access Skeleton Query

All of the preceding queries have been directed to only one topic or one CH node and have only requested that one instance be returned. Some queries require a sub-hierarchy to be accessed, such as, "What is the appearance of Clyde?". This requires the appearance sub-hierarchy to be accessed. Others require all instances to be returned or a combination of the above. For example, "What are all the animals you know?". To allow this two flags have been added to the basic query form. They are a "sub-hierarchy" flag and a "many" flag. The "sub-hierarchy" flag says to use the sub-hierarchy rooted at the node sought to access propositions rather than just that node. The "many" flag says to return

```
?sub-hier many [?x ?y{tp.gen:mammal}]
```

What are all the instances of things that might be mammals you know?

```
?s m [Clyde-head ?y{tp.physical-relation} ?x]
```

What are all of the physical relationships with Clydes's head that you know?

Figure 6.4 Query Flags

as many instances as possible. Admittedly these are a little crude, but they are only intended to show sub-hierarchy accessing and accessing of all matching

propositions. The flags[1] are inserted between the '?' and the query proposition. Figure 6.4 contains some examples of these. The difference between the first query of figure 6.4 and the query

```
?s m [?x mammal]
```

is not obvious. Both access all individuals that have been classified in the sub-hierarchy rooted at the 'mammal' node, but the first pattern matches all of their generalization propositions (e.g., [Clyde elephant], [Tom wolf]) since its predicate is a query variable, whereas the second pattern matches only those generalization propositions that have 'mammal' as their predicate. Thus, the first pattern returns all individuals classified under the sub-hierarchy rooted at the 'mammal' node and the second pattern returns only those individuals that have been explicitly predicated as being mammals. The difference in the propositions matched is evident from the output of the query answerer. A typical response to the first query would be

```
[Clyde elephant]. Need to show: [Clyde mammal],
```

clearly indicating that 'Clyde' is merely a candidate 'mammal'. A typical response to the second query would be

```
[Harry mammal],
```

assuming that 'Harry' had been explicitly predicated as a mammal.

---

[1] As with all flags and keywords in this system they may be abbreviated to their shortest unique initial sub-string.

Some sample queries, encompassing the types of queries discussed in this section, were input to the system. The propositions in the knowledge base, the hierarchy in use and the result of these queries are shown in appendix G.

6.2.2.2 Query Answerer

This section describes how the query answerer works. The query is parsed using the same parser as for propositions with the extensions of query variables in argument and predicate positions. The query is classified using the same classifier as the knowledge input routines use. A classification involving a non-query variable is scanned for in the classification. If one is found then depending on the "sub-hierarchy" flag, the propositions attached to the sub-hierarchy or node of the access skeleton indicated are accessed. Then each proposition is matched in turn against the query (pattern). If one does match then depending on the "many" flag, either that proposition plus bindings for the query variables is returned or the process continues on to find as many matching propositions as possible. If no propositions are matched or the "many" flag was specified then the generalization routines are brought into play.

The generalization routines allow generic information to be used to answer a question. For example, to answer the question "Is Clyde's head joined to his neck?" when the

information that this is so is not stored with the concept 'Clyde' but with a generalization of 'Clyde', say, the 'mammal' concept. There are two different routines. One handles generalizations of independent concepts, such as 'Clyde' to 'elephant', and the other handles rematching of dependent concepts, such as 'Clyde-head' (Clyde's head) to the elephant's head. The first uses the TAS attached to the independent concept to find all generalizations of it. The second uses the function tables and TASs associated with independent concepts to do the rematching of corresponding dependent concepts.

?[Clyde ?x{tp.colouring}]

generalizes to

?[[¬[c.1 elephant]]|[c.1 ?x{tp.colouring}]]

assuming that c.1 is the universal quantifier for elephant in the main net.

?[John believes [Clyde ?x{tp.colouring}] ?y]

generalizes to

?[John believes [[¬[c.2 elephant]]|
[c.2 ?x{tp.colouring}] ?y]

assuming that c.2 is the universal quantifier for elephant in John's subnet. Note that the strength of John's belief is not specified but is asked for in this case by query variable '?y'.

Figure 6.5 Generalizations of Independent Concept Queries

It is necessary to change the form of the query if the

generalization goes from an instance (e.g., Clyde) to a generic concept (e.g., elephant). The form is changed from a simple proposition to an implication with the generic concept with its universal argument as antecedent and the simple proposition as consequent. Figure 6.5 shows two generalizations, one in the main net, the other in a subnet.

?[Clyde-head joined-to ?y]

generalizes to

?[[¬[c.1 elephant]]|[c.9 joined-to ?y]]

generalizes to

?[[¬[c.4 mammal]]|[c.10 joined-to ?y]]

assuming that c.4 is the universal concept associated with the 'mammal' concept and that c.9 and c.10 are the generic elephant's head and the generic mammal's head respectively.

Figure 6.6 Generalizations of Dependent Concept Queries

A query that is classified with respect to a dependent concept, and some generalizations of the query are shown in figure 6.6. The concepts c.9 and c.10 in figure 6.6 are found by looking up the function name of 'Clyde-head', say 'f', and the argument of the function, namely Clyde, and then ascending in the generalization hierarchy from Clyde. At each generalization of Clyde a value of 'f' is sought. If one is found then the generalized query is looked for in the propositions classified under the value of 'f' found. Looking for corresponding dependent concepts works even if

'f' does not have a value for some generalization of Clyde. That generalization is skipped and the next one up is examined. Since only function values are being sought no disruption of the generalization chain occurs. Note that the classification is not done for each generalization since it stays the same for successive corresponding dependent concepts.

Once a matching proposition has been found a positive reply can be output. The reply given depends on the form of the query. If no query variables were part of the query then a "yes" is output. If a query variable is part of the query then a substitution is attempted. If all substitutions are successful the proposition formed is printed as the answer. For example, the output for the first query in figure 6.1 is "yes" and the output for the first query of figure 6.2 is the proposition

[Clyde grey].

Substitutions follow the following rules. If the query variable is a predicate then a straight substitution of the predicate bound to the query variable suffices. If the query variable is bound to a dependent concept then the corresponding dependent concept associated with the individual that was generalized is sought using the function table attached to that individual. If it is found then a substitution is performed, otherwise a message is printed which states that the query variable was bound to a concept but no corresponding dependent concept associated with the

individual generalized was found. For example, the latter case would occur if the system only knew that Tom was an elephant and the question "Does something exist that is Tom's head?" was asked. Tom's head has not been mentioned so it would not exist in the knowledge base, but the knowledge that all elephants have a head is available. Thus the system would answer yes Tom has a head, but I do not know it. If the query variable is bound to a universal concept then the individual generalized is substituted for the query variable. This only works for the restricted queries permitted by this system. If the query variable is bound to an independent existential concept then a straight substitution of bound concept for the query variable is performed.

Chapter 7

CONCLUSIONS

7.1 Results

The first problem examined by this thesis was the organization of groups of propositions that form various conceptions of the world. The solution obtained was to place each group of propositions into a subnet of its own. The problem of providing attachment points for the organizational structures within a subnet and the problem of cross identifying individuals between nets was solved by using virtual concepts inside subnets. The structure of subnets and the main net were made logically the same so that the same organizational methods can be used in all nets. It was shown that with this organization, propositions within a conception of the world, a subnet, can be accessed as easily and in the same fashion as propositions in the main net. Another result of using virtual concepts only as attachment points for organizational structures is that propositions are constructed from one common set of concepts, thus allowing the detection and removal of duplicate sentential forms. It was shown that the removal of duplicate sentential forms is advantageous. A partial method to do this was proposed and implemented.

Another problem examined by this thesis was that of associatively accessing individuals. This problem is particulary evident in subnets where an explicit sub/super concept hierarchy is often not present. This was solved by the superposition of a global concept hierarchy on the knowledge base and the addition of a concept access skeleton to each net. Concept access skeletons allow associative access within each net to concepts of a particular type. The concepts found are treated as mere candidates because the explicit sub/super concept hierarchy in a subnet may not agree with the global concept hierarchy.

Several improvements to access skeletons were introduced. They are new access skeleton building and access algorithms and a new access skeleton structure to support the new algorithms. The building and access algorthims were shown to be more efficient in time than previous realizations. The new structure was also shown to be more efficient in space than Goebel's <1977> structure.

An alteration was made to the definition of topic indicator links so that argument position plays a role in determining what topic a particular predicate specifies. This was shown to be important to allow proper classification of propositions that have n-ary (n>1) predicates as constituents.

The implementation shows the use of function tables,

concept access skeletons, and topic access skeletons in the main net and in subnets to answer some simple queries.

7.2 Future Research

Property and relationship inheritance using function tables requires that the same Skolem function name be used for corresponding dependent concepts. An automated method for matching corresponding dependent concepts and thus unifying Skolem function names needs to be developed. A solution to this would seem to incorporate a solution to the problem left open by the proposed partial solution to the problem of avoiding duplicate sentential forms. The proposed solution left open the question of deciding when two existential variables are "the same".

There are a few questions left unanswered by the solution to the property and relationship inheritance problem. One of them is "Should dependent concepts have an index to their parts?". For example, should the hand dependent on the generic human have an index to the fingers? Another question is "How should the dependent concepts of a generic part be coordinated with corresponding dependent concepts of dependent parts?". For example, how should the fingers of the generic hand be related to the fingers of the hand dependent on the generic human?

Partitionings were briefly mentioned in the section dealing with duplicate sentential forms. Schubert <1979>

proposed methods for operating on partitionings, but a representational structure for them which allows efficient performance of these operations needs to be defined.

No method was proposed to organize episodic knowledge. Such an organization is needed since planning routines and inferences involving stories or events in the "real world" will utilize causal and time dependencies quite heavily.

The concept hierarchy allows individuals to be accessed only via their known types. This should be extended to other types of predications, such as attributes (e.g., grey, woodsman), relationships (e.g., likes, gives) and relative attributes (e.g., large, thin). A combination of a topic hierarchy with concept hierarchies at its leaves may be the solution to this problem.

Another area that should be examined is the query answerer. A more powerful query answerer would better explore the possibilities of the organizational structures, of the credibility propositions and of the degrees of belief attached to propositional attitude propositions. The use of different conceptions of the world should be given special attention since it can be used to infer a person's beliefs, such as, inferring John's political beliefs once it is known that he is a liberal. One place for extension in the query answerer is in the pattern matching component. At present no partial matches, excepting query variables, are allowed. Partial matches are needed to answer questions such as "What

colour are sheep?", when it is known that sheep are black or white. The literals of this query would only match some of the literals of the proposition in the knowledge base. This sort of matching is a straightforward extension of the current matching capabilities of the system. Partial matching would also entail an extension in the part of the query answerer that interprets the results of the matching process.

References

Bobrow, D. G. and Winograd, T. (1977): "An Overview of KRL, a Knowledge Representation Language", Cognitive Science 1, 3-46.

Cercone, N. (1975): "The Nature and Computational Use of a Meaning Representation for Word Concepts", Technical Report TR75-9, U. of Alberta, Edmonton, Canada, July.

Cercone, N. and Schubert L. K. (1975): "Toward a State Based Conceptual Representation", Advance Papers of IJCAI 4, Tblisi, U.S.S.R., September 3-8, 83-90.

Creary, L. G. (1979): "Propositional Attitudes: Fregean Representation and Simulative Reasoning", Proc. IJCAI 6, Tokyo, Japan, August 20-23, 176-181.

Fikes, R. and Hendrix, G. G. (1977): A Network-Based Knowledge Representation and Its Natural Deduction System", Proc. IJCAI 5, MIT, Cambridge, Mass., U.S.A., August 22-25, 235-246.

Goebel, R. (1977): "Organizing Factual Knowledge in a Semantic Network", M.Sc. Thesis, Dept. of Comp. Sci., U. of Alberta, Edmonton, Canada, September.

Hayes, Philip (1977): "On Semantic Nets, Frames and Associations", Proc. IJCAI 5, MIT, Cambridge, Mass., U.S.A., August 22-25, 99-107.

Hendrix, G. (1975): "Expanding the Utility of Semantic Networks Through Partitionings", Advance Papers of IJCAI 4, Tblisi, U.S.S.R., September 3-8, 115-121.

Hendrix, G. G. (1979): "Encoding Knowledge in Partitioned Networks", Associative Networks - The Representation and Use of Knowledge by Computers, Findler N. V. (ed.), Academic Press, 51-92.

Hintikka, J. (1971): "Semantics for Propositional Attitudes", Reference and Modality, Linsky, L. (ed.), Oxford University Press, London, England, 145-167.

Hughes, G. E. and Cresswell, M. J. (1974): An Introduction to Modal Logic, Methuen and Co. Ltd., London, England.

Kling, R. (1973): "FUZZY PLANNER: Computing Inexactness in Procedural Problem-Solving Languages", Technical Report 168, Comp. Sci. Dept., U. of Wisconsin, Madison, Wisc., U.S.A., February.

LeFaivre, R. A. (1974): "The Representation of Fuzzy Knowledge", Journal of Cybernetics 4 #2, 57-66.

LeFaivre, R. A. (1976): "Procedural Representation in a Fuzzy Problem-Solving Systems", AFIPS Conf. Proc. 1976, 1069-1074.

McCarthy, J. (1977): "Epistemological Problems of Artificial Intelligence", Proc. IJCAI 5, Cambridge, Mass., U.S.A., August 22-25, 1038-1044.

Moore, R. C. (1975): "A Serial Scheme for the Inheritence of Properties", ACM SIGART Newsletter 53, August, 8-9.

Moore, R. C. (1977): "Reasoning About Knowledge and Action", Proc. IJCAI 5, Cambridge, Mass., U.S.A., August 22-25, 223-227.

Quillian, M. R. (1968): "Semantic Memory", Semantic Information Processing, Minsky, M. (ed.), MIT Press, Cambridge, Mass., U.S.A., 227-270.

Quillian, M. R. (1969): "The Teachable Language Comprehender: A Simulation Program and Theory of Language", Comm. ACM 12 #8, 459-475.

Quine, W. V. O. (1971a): "Quantifiers and Propositional Attitudes", Reference and Modality, Linksy, L. (ed.), Oxford University Press, London, England, 101-111.

Quine, W. V. O. (1971b): "Reference and Modality", Reference and Modality, Linsky, L. (ed.), Oxford University Press, London, England, 17-34.

Raphael, B. (1968): "SIR: A Computer Program for Semantic Information Retrieval", Semantic Information Processing, Minsky, M. (ed.), MIT Press, Cambridge, Mass., U.S.A., 33-145.

Reder, L. M. and Anderson, J. R. (1979): "Use of Thematic Information to Speed Search of Semantic Nets", Proc. IJCAI 6, Tokyo, Japan, August 20-23, 708-710.

Rychener, M. D. (1979): "A Semantic Network of Production Rules in a System for Describing Computer Structures", Proc. IJCAI 6, Tokyo, Japan, August 20-23, 738-743.

Schank, R. C. (1973): "Conceptual Dependencies", Computer Models of Thought and Language, Schank, R. C. and Colby, K. M. (ed.s), W. H. Freeman and Co., San Francisco, U.S.A., 187-247.

Schank, R. C. and Rieger, C. J. III (1974): "Inference and the Computer Understanding of Natural Language",

Artificial Intelligence 5, 373-412.

Schubert, L. K. (1975): "Extending the Expressive Power of Semantic Networks", Advance Papers of IJCAI 4, Tblisi, U.S.S.R., September 3-8, 158-164.

Schubert, L. K. (1978): Personal communication.

Schubert, L. K. (1979): "Problems with Parts", Proc. IJCAI 6, Tokyo, Japan, August 20-23, 778-784.

Schubert, L. K., Goebel, R. and Cercone, N. J. (1979): "The Structure and Organization of a Semantic Net for Comprehension and Inference", Associative Networks - The Representation and Use of Knowledge by Computers, Findler N. V. (ed.), Academic Press, 121-175.

Shapiro, S. C. (1971): "A Net Structure for Semantic Information Storage, Deduction, and Retrieval", Advance Papers of IJCAI 2, Imperial College, London, England, September 1-3, 512-523.

Zadeh, L. A. (1975): "The Concept of a Linguistic Variable and Its Application to Approximate Reasoning - I and II", Infor. Science 8, 199-249, 301-357.

APPENDIX A

Access Skeleton Building Algorithm

This appendix describes the algorithm used to add a new access skeleton node, N, to a path contracted access skeleton, PCAS, where the access skeleton is built from a tree structured hierarchy. There are four basic cases to consider.

1) If the PCAS is empty then N becomes the PCAS.

2) If N is an ancestor of the top node of the PCAS then
   make the root node of the PCAS a descendant of N,
   N becomes the new root of the PCAS.

3) If the root of the PCAS is an ancestor of N then
   find the deepest ancestor, DA, (one furthest from the root of the PCAS) of N in the PCAS.

   a) if N lies between DA and one of DA's descendants, D, then
      add N between DA and D. Because of the construction of the PCAS, D will be the only descendant of N that is also a descendant of DA.

   b) if N and one of the descendants of DA, D, have a common ancestor in the hierarchy that is below DA then
      find the deepest common ancestor, DCA, between D and N, add DCA to the PCAS using this method recursively,
      add N as DCA's descendant.

   c) if N has no relationships with DA's descendants other than DA then
      add N as a descendant of DA.

4) if N and the root of the PCAS are not directly related then
   find the deepest common ancestor, DCA, of N and the root of the PCAS in the hierarchy,
   add DCA to the PCAS (type 2 addition),
   add N as a descendant of DCA (type 3c addition).

This algorithm ensures that the hierarchical properties of the hierarchy are maintained in an access skeleton. This basically requires that no fork nodes, nodes that have two paths leaving them in the access skeleton, are deleted or not entered in the access skeleton when a new node is added that makes them into a fork node.

APPENDIX B

Proposition Input Grammar

Meta Syntax

```
, - alternation.
[ ... ] - ... is optional.
* - previous entity any number of times including 0
: - separates left hand side from right hand side of
     grammar rule.
; - rule terminator.
'...' - characters in apostrophies are taken literally.
< ... > - ... is an English description of a terminal.
/* ... */ - ... is a comment.
```

Grammar

```
sent : [ prop ]* < end of line >
     ;
prop : [ quant ]* '[' arg pred [ timeargs ] [ arg ]* ']'
     , logical-prop ;
     ;
quant : 'A*' qident /* universal quantification */
      , 'E*' qident /* existential quantification */
      ;
qident : ident
       ;
arg : prop
    , concept
    , qident
    , function
    ;
timeargs : '<' arg [ ',' arg ] '>'
         ;
concept : ident
        ;
pred : arg
     ;
logical-prop : monadic-ex
             , dyadic-ex
             , polyadic-ex
             ;
monadic-ex : '[' monadic-op prop ']'
           ;
monadic-op : '¬' /* not */
           , '#' /* necessarily */
           ;
dyadic-ex : '[' prop '=>' prop ']' /* implication */
          ;
polyadic-ex : '[' prop '&' prop [ '&' prop ]* ']'
               /* n-ary conjunction */
            , '[' prop '|' prop [ '|' prop ]* ']'
```

```
                    /* n-ary disjunction */
                  ;
function : '(' function-name arg [ arg ]* ')'
         ;
function-name : ident
              ;
ident : firstchar [ otherchars ]*
      ;
firstchar : < upper and lower case alphabet >
          , < digits >
          , '.'
          ;
otherchars : firstchar
           , '-'
           , '_'
           ;
```

Note: a line may be broken across several physical lines by "escaping" the 'end of line' character with a backslash, "\".

APPENDIX C

Structure Building Commands

This appendix describes the grammar and some of the semantics of the commands which build the super-hierarchy and add indicator links from concepts to hierarchy nodes. The grammar uses some non-terminals from appendix B. One additional piece of meta syntax is defined. It is:
"..." - ... may be entered as any initial sub-string different from other strings in double quotes (e.g., "hier" could be entered as "h", "hi", "hie", or "hier" so long as no other command starts with "a" or "ad"). If the sub-string entered is an intial sub-string of more than one acceptable string then one of the acceptable strings is chosen.

Grammar

```
bldcmd : "hier" topic "marked-by" [ concept [ argno [
          relevance ] ] [ ',' concept [ argno [ relevance ]
          ] ]* ] < end of line >
       , "hier" topic "super" [ topic [ relevance ] [ ','
          topic [ relevance ] ]* ] < end of line >
       ;
```

The first form builds indicator links. The second form builds the super-hierarchy. 'Relevance', 'topic' and 'argno' are identifiers. 'Argno' is interpeted as the argument number of concept to which the indicator link applies. 'Relevance' specifies the relevance of the link being made. It is taken to be a concept name.

APPENDIX D

Structure Modification Commands

This appendix describes the grammar and some of the semantics of the commands used to modify Skolem function tables. The grammar uses some of the non-terminals from appendix B.

Grammar

```
modcmd : "add" [ subnet ]* [ ',' ] arg-concept fcn-name
          value-concept < end of line >
       , "change" [ subnet ]* [ ',' ] concept old-fcn-name
          new-fcn-name < end of line >
       ;
subnet : "subnet" concept
       ;
          This construct means "move to the subnet attached
          to 'concept'".
```

All of 'arg-concept', 'fcn-name', 'value-concept', 'old-fcn-name' and 'new-fcn-name' are simply identifiers as in appendix B, but checks are made to ensure they fall into the classes indicated by their names. The commands "add" and "change" respectively add and change function names in function tables attached to concepts.

APPENDIX E

Utility Commands

This appendix describes the grammar and some of the semantics of the utility commands available under the system. The grammar uses the meta syntax and some of the non-terminals from the previous grammars.

Grammar

```
utlcmds : "print" "hier" < end of line >
        , "print" "prop" [ prop-number ]* < end of line>
        , "print" [ subnet ]* "conc" [ conc-name ]* < end of
          line >
        , "print" [ subnet ]* "access-skeleton" < end of
          line >
        , "print" "all" < end of line >
        , "print" "number" [ number ]* < end of line >
          The print command allows the internal structures
          to be examined. Propositions do not have a name
          so the relative offset in memory must be specified
          to specify a proposition. The relative offset in
          memory can be obtained by printing a concept under
          which the proposition has been classified. If no
          numbers are specified after the "prop" keyword all
          propositions in the knowledge base are printed.
          Propositions are "pretty printed" using this
          command. If the internal structure is of more
          interest than the proposition, then the "number"
          sub-command should be used. The "conc" sub-
          command expects a list of concept names. If a
          null list is given all concepts referenced by the
          net specified by the "subnet" modifier are
          printed. The "access-skeleton" sub-command prints
          the concept access skeleton associated with the
          net specified by the "subnet" modifier. The "all"
          sub-command prints all entries in the sentential
          form hash table. The "number" sub-command prints
          the contents of memory specified by the relative
          offset 'number'.
        , "file" file-name < end of line >
        , "save" [ file-name ] < end of line >
        , "restore" [ file-name ] < end of line >
          These commands allow input to be read from a file,
          the knowledge base to be saved on a file and the
          knowledge base to be restored from a file. The
          last two have a default file name of 'pdb.save'.
        , "set" "echo" value < end of line >
        , "set" "match-stat" value < end of line >
        ;
value : "on"
      , "off"
```

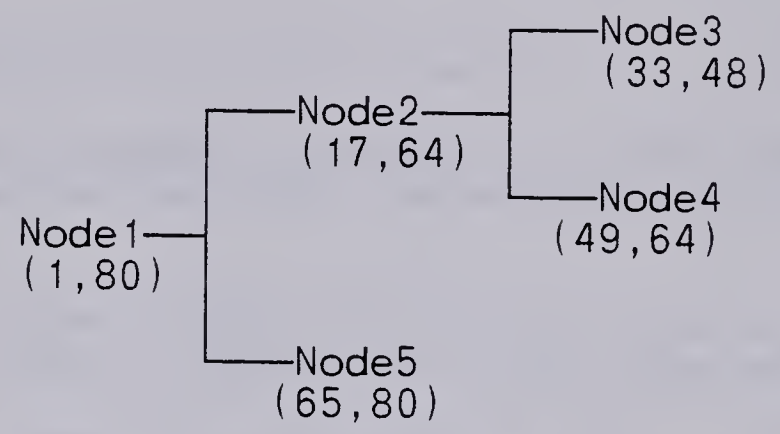
, < number >
;

The set commands allows some global flags to be turned off (0) or on (1) or to be set to any number. The flags "echo" and "match-stat" can only take on the values off or on. The "echo" flag controls echoing to the terminal when input is coming from a file. The "match-stat" flag controls collecting and printing of match statistics when the query sub-system is being used.

APPENDIX F

Hierarchy Numbering Algorithm

This appendix contains a simple scheme to reduce the number of renumberings of the hierarchy required. The actual renumbering scheme is O(n) where n is the number of nodes in the hierarchy. This scheme leaves "holes" in the



Hierarchy before addition of 'A'

Hierarchy after addition of 'A'

Figure F.1 Addition of a Node to a Hierarchy

numbering of the hierarchy. The "holes" can be used to number nodes that are added to the hierarchy without requiring a renumbering of the hierarchy. The holes are left between a node and its first offspring (in depth first order) if it has any or at the node if is a leaf node. Figure F.1 shows a hierarchy numbered with this scheme and the same hierarchy with a node, 'A', added. Note, the addition of 'A' did not require renumbering the hierarchy, but just the redistribution of the "hole" at Node2. The numbering algorithm is as follows. A global indentification number is required to keep track of the number of nodes examined and the space left for holes. It is initialized to

zero. The routine is initially called with the root of the hierarchy as node.

1) add one to global identification number to get next unused one.

2) set identification number of node to global identification number.

3) leave room for new nodes by adding the size of the desired hole, say 15, to the global identification number.

4) for all offspring of node execute this routine.

5) set "last descendant" of node to global identification number to complete the descendent bracket for node.

The algorithm to add a new node is given below. Before it is given a few variable definitions are needed. They are:

newnode - new hierarchy node to add to hierarchy.
node - hierarchy node which is newnode's parent.
span - span of unused numbers associated with the hierarchy node node.

The algorithm is as follows:

1) calculate the size of the "hole", span, associated with node. This is done as follows.
if node is a leaf then
    span is the width of the descendent bracket of node
else
    span is the identification number of node's first offspring - 1 - the identification number of node
fi

2) if there is a hole (i.e. span > 0) then
    split the hole between node and newnode. Newnode's descendent bracket is set up as follows.
    Newnode's identification number becomes node's identification number + (span+1)/2.
    Newnode's last descendent number becomes
    if node is a leaf then
        node's last descendent number
    else
        node's first offspring's identification number - 1
    fi
else
    renumber hierarchy
fi

APPENDIX G

Sample Run

This appendix contains a list of the propositions input to the system and an annotated sample run of the implementation. Comments are indented and delineated by blank lines. Input to the system is prefixed by a '*'. In cases where the output is lengthy, it has been shortened, but the salient features of it have not been removed. The propositions that were entered into the system are:

```
A*x E*y E*z E*u[[x higher-animal]=>[[y head-of x] &\
[z neck-of x] & [z joined-to y] & [u body-of x] &\
[y joined-to z] & [u joined-to z] & [z joined-to u]]]

A*x[[x mammal]=>[x higher-animal]]
A*x E*y E*z E*t E*m[[x mammal]=>[[x higher-animal] &\
[y nose-of x] & [y attached-to z] & [z head-of x] &\
[m mouth-of y] & [t teeth-of m] & [x hairy] & [x warm] &\
[t inside m] & [y attached-to z]]]

A*x[[x elephant]=>[[x mammal] & [x grey] & [x big]]]
A*x E*y E*z[[x elephant]=>[[y head-of x] & [z trunk-of x] &\
[z nose-of y] & [z long] & [z prehensile]]]
A*x [[x elephant]=>[[x likes Bruce] & [Bruce likes x]]]
A*x A*y[[[x elephant] & [y peanut]]=>[x likes y]]
A*x A*y[[[x elephant] & [y mouse]]=>[y likes x]]

A*x[[x whale]=>[[x mammal] & [x blue]]]

A*x[[x man]=>[x mammal]]

A*x[[x bird]=>[x higher-animal]]
A*x[[x thrush]=>[x bird]]
A*x E*y E*z[[x robin]=>[[x thrush] & [y breast-of x] &\
[y red] & [z head-of x] & [[z black]|[z grey]]]]

A*x [[x wolf]=>[x mammal]]
A*x E*t E*m[[x wolf]=>[[t teeth-of m] & [m mouth-of x] &\
[m large] & [t sharp] & [t pointy] & [x grey] & [t white] &\
[m red]]]

A*x [[x mouse]=>[x mammal]]
A*x E*y[[x mouse]=>[[y tail-of x] & [y long] & [y thin] &\
[y hairless]]]
A*x[[x mouse]=>[x small]]

[Bruce man]

[Nibor robin]
[Nhead head-of Nibor]
```

```
[Nbreast breast-of Nibor]

[Clyde elephant]
[Chead head-of Clyde]
[Cneck neck-of Clyde]
[Cnose trunk-of Chead]
[Cbody body-of Clyde]
[Ctail tail-of Cbody]

[Tom elephant]

[Hilda mouse]
[Htail tail-of Hilda]
[Hilda likes Tom]

[Spot whale]

[Diane wolf]
[Dhead head-of Diane]
[Dteeth teeth-of Dmouth]
[Dnose nose-of Dhead]
[Dtail tail-of Diane]
[Dmouth mouth-of Dhead]
[Dneck neck-of Diane]
[Dbody body-of Diane]

[Aaron wolf]
[Aaron loves Diane]
[Diane loves Aaron]

[Nibor on-top-of<t1,t2> Clyde]
[Clyde supports<t1,t2> Nibor]
[Diane eat<t3,t4> Nibor]
[t2 less-than t3]

[LRRH is-a-story-in-which E*x E*y E*w E*z [[x wolf] &\
[y girl ] & [z woman] & [w man] & [w woodsman] &\
[z grandmother-of y] & [x eat<t5,t6> z] &\
[x eat<t7,t8> y] & [w kill<t9,t10> x] & [t6 less-than t7] &\
[t8 less-than t9]]]

[John thinks A*x [[x elephant]=>[x mammal]] 1.0]
[John believes A*x E*y E*z [[x mammal]=>[[y head-of x] &\
[z neck-of x] & [y joined-to z] & [z joined-to y]]]\
.98]
[John believes [Spot whale] .99]
[John thinks [Tom elephant] 1.0]
[John believes [[Thead head-of Tom] &\
[Tneck neck-of Tom]] .9]
E*y [John believes A*x[[x whale]=>[[x fish] & [x blue] &\
[x large]]] y]
[John imagines A*x[[x elephant]=>[[x black] & [x small]]]\
.8]
```

The hierarchy, indicator links, propositions and function table modifications have been previously entered into the system. The resultant knowledge base was stored in a file "save.link". Thus, the first command of the sample run is to restore the file "save.link". There is only one notation convention followed in the sample run. It is that topic node names start with the characters "tp." and are often abbreviated (e.g., the 'generalization' topic is named "tp.gen"). The following is the sample run. "ready" is the initial system prompt.

```
ready
* res save.link
* print pro
```

Print all propositions (clauses) in the knowledge base. Only a few are shown here. There are actually 108 clauses generated by the above data after normalization.

```
[Aaron loves Diane]
[Diane loves Aaron]
[Clyde supports<t1,t2> Nibor]
[[¬ [c.22 wolf]] | [(f.7 c.22) sharp]]
[John believes [[¬ [c.31 mammal]] | [(f.16 c.31) joined-to
(f.15 c.31)]] .98]
[Dmouth mouth-of Dhead]
[[¬ [c.1 higher-animal]] | [(f.1 c.1) joined-to (f.2 c.1)]]
[LRRH is-a-story-in-which [c.27 kill<t9,t10> c.29]]
[John believes [[¬ [c.35 whale]] | [c.35 large]] c.34]

* p h
```

This command prints the current hierarchy. Each level in the hierarchy is indented two spaces further right than the previous level.

```
top_of_hierarchy
  tp.phys-qual
    tp.substance
    tp.mass
    tp.ext-qual
      tp.tact-qual
        tp.resil
        tp.hard
        tp.tex2
      tp.odour
      tp.appearance
        tp.tex1
        tp.trans
        tp.col
```

```
        tp.form
    tp.size
tp.episode
tp.behav
    tp.development
    tp.social
        tp.play
        tp.fighting
        tp.mating
        tp.ritual
        tp.communication
    tp.constructive-act
    tp.self-main
        tp.sleep
        tp.cleaning
        tp.elimin
        tp.feed
    tp.locamotion
tp.func
    tp.passive-func
    tp.occupation
tp.stat-rel
    tp.abstract-rel
        tp.arith
        tp.membership
        tp.ownership
        tp.control
        tp.kinship
    tp.emot-rel
        tp.emot-receip
        tp.emot-giver
    tp.phys-rel
        tp.affinity
        tp.force
        tp.containment
        tp.part-of
            tp.has-part
            tp.is-part
        tp.location
tp.men-qual
    tp.int-disp
    tp.emot-disp
tp.topics
tp.spec
    plant
        peanut
        tree
    creature
        bird
            kestrel
            raven
            crow
            bat
            sparrow
```

```
      robin
    reptile
      lizard
      snake
    fish
      walleye
      pike
      gar
    mammal
      wolf
      whale
      cat
      mouse
      elephant
  human
    boy
    girl
    man
    woman
tp.gen
tp.prop_att
  tp.emot-prop-att
    tp.look-forward-to
    tp.regret
    tp.fear
    tp.hope
  tp.intention
    tp.belief
```

The next three print commands show the structures attached to an independent concept, a universal concept, and a dependent concept. Note the size of the TASs attached to the existential concepts compared to the size of the hierarchy in the system.

```
* p c wolf
Conc1322 Ext wolf
Indicator links
tp.gen

Access Skeleton
  top_of_hierarchy
    tp.has-part Prop10819
    tp.col Prop10975
    tp.gen Prop6052


Uni List
Conc10552 Uni c.22


* p c c.22
```

```
Conc10552 Uni c.22
Function table
  Index f.6
  ObjeK Conc11248 Ext c.24
  Index f.7
  ObjeK Conc11206 Ext c.23

Controlling Type Predicate
Conc1322 Ext wolf


* p c c.24
Conc11248 Ext c.24
Scope inclusion
  Binder
    f.6
  Bindee
    Conc10552 Uni c.22

Access Skeleton
  top_of_hierarchy
    tp.part-of
      tp.has-part Prop10783
      tp.is-part Prop10819
    tp.phys-qual
      tp.size Prop10858
      tp.col Prop11053
```

The next two print commands show the internal structure of two sentential forms. The first is a clause and the second is a sentential form used in many clauses as shown by the length of the backlist.

```
* p n 10783
Prop10783 time 14
Logical or
Arg1 Prop10538 time 13
  Logical not
  Arg1 Prop9720 time 13
    Pred Conc1322 Ext wolf
    Arg1 Conc10552 Uni c.22
Arg2 Prop10636 time 14
  Pred Conc4245 Ext teeth-of
  Arg1 Conc11206 Ext c.23
  Arg2 Conc11248 Ext c.24


* p n 10538
Prop10538 time 13
Backlist
  Prop10783 time 14
  Prop10819 time 14
```

```
  Prop10858 time 14
  Prop10897 time 14
  Prop10936 time 14
  Prop10975 time 14
  Prop11014 time 14
  Prop11053 time 14
  Prop6052 time 13
Logical not
Arg1 Prop9720 time 13
  Pred Conc1322 Ext wolf
  Arg1 Conc10552 Uni c.22


* p p 10783 10819 10858 10897 10936 10975 11014 11053 6052
```

This command prints all the clauses which contain the above sentential form, Prop10538.

```
[[¬ [c.22 wolf]]  | [(f.7 c.22) teeth-of (f.6 c.22)]]
[[¬ [c.22 wolf]] |  [(f.6 c.22) mouth-of c.22]]
[[¬ [c.22 wolf]]  | [(f.6 c.22) large]]
[[¬ [c.22 wolf]] |  [(f.7 c.22) sharp]]
[[¬ [c.22 wolf]]  | [(f.7 c.22) pointy]]
[[¬ [c.22 wolf]] |  [c.22 grey]]
[[¬ [c.22 wolf]]  | [(f.7 c.22) white]]
[[¬ [c.22 wolf]] |  [(f.6 c.22) red]]
[[¬ [c.22 wolf]]  | [c.22 mammal]]
```

The next two concepts show the presence of relevance and argument number on their indicator links.

```
* p c clear teeth-of
Conc3809 Ext clear
Indicator links
tp.col Relevance Conc3840 Ext .8
tp.trans


Conc4245 Ext teeth-of
Indicator links
tp.has-part Arg# 2 tp.is-part Arg# 1
```

Now for the query answering. The statistics printed for each query ("match-stat" flag is on) are the number of propositions examined and the number of generalizations made while trying to answer the query.

```
* set mat on
* file querys
```

This query and the next two use the CAS associated with the main net to find entities that may fullfil the requirements.

```
* ?sub-hier many[?x ?y{tp.gen:mammal}]
```

What entities may be mammals?

```
# props looked at 6.  # generaliztions 0.
[Aaron wolf].  Need to show: [Aaron mammal]
[Diane wolf].  Need to show: [Diane mammal]
[Spot whale].  Need to show: [Spot mammal]
[Hilda mouse].  Need to show: [Hilda mammal]
[Tom elephant].  Need to show: [Tom mammal]
[Clyde elephant].  Need to show: [Clyde mammal]

* ?s m[?x mammal]
```

What entities are explicitly predicated to be mammals?

```
# props looked at 6.  # generaliztions 0.
Nothing found

* ?m[?x wolf]
```

What entities are predicated to be wolves?

```
# props looked at 2.  # generaliztions 0.
[Aaron wolf]
[Diane wolf]

* ?[Diane mammal]
```

Is Diane a mammal? To answer this requires looking at the generalizations of wolf, the generalization of Diane, which in turn requires accessing the generalization node the TAS of Diane and the TAS of the generic wolf.

```
# props looked at 2.  # generaliztions 1.
Yes

* ?[Clyde mammal]
```

This is similar to the above query.

```
# props looked at 2.  # generaliztions 1.
Yes

* ?[Diane ?x{tp.col}]
```

This query uses a query variable in the predicate position. Note the specification of the topic of inquery, tp.col (colouring). What colour is Diane?

```
# props looked at 1.  # generaliztions 1.
[Diane grey]

* ?s m[Diane ?x{tp.appearance}]
```

A more general query, that requires the use of the "sub-hierarchy" and "many" flags. What is Diane's appearance?

```
# props looked at 2.  # generaliztions 3.
[Diane hairy]
[Diane grey]

* ?s m[?prt ?pred{tp.part-of} Diane]
```

Find all the things that have been predicated as part of Diane or one of her generalizations. Some of the parts appear more than once. This is due to corresponding parts being predicated as part of Diane, part of the generic wolf and part of the generic mammal.

```
# props looked at 10.  # generaliztions 3.
[Dhead head-of Diane]
[Dneck neck-of Diane]
[Dbody body-of Diane]
[Dnose nose-of Diane]
[Dhead head-of Diane]
[Dmouth mouth-of Diane]
[Dbody body-of Diane]
[Dneck neck-of Diane]
[Dtail tail-of Diane]
[Dhead head-of Diane]
```

The next three queries extract information about dependent concepts. Function tables are used to find corresponding dependent concepts.

The first two queries show the effect of the "sub-hierarchy" flag. The first query looks at just those propositions classified under 'tp.phys-rel', whereas the second query looks at the propositions classified in the sub-hierarchy attached to 'tp.phys-rel'.

```
* ?m[?y ?x{tp.phys-rel} Dhead]
# props looked at 3.  # generaliztions 3.
[Dneck joined-to Dhead]
[Dnose attached-to Dhead]

* ?s m[?y ?x{tp.phys-rel} Dhead]
# props looked at 8.  # generaliztions 3.
[Dneck joined-to Dhead]
[Dnose attached-to Dhead]
```

```
[Dmouth mouth-of Dhead]
[Dnose nose-of Dhead]

* ?s m[Dteeth ?x{tp.appearance}]
# props looked at 2.  # generaliztions 3.
[Dteeth pointy]
[Dteeth white]
```

The next three queries extract the short "episode" in the knowledge base. The episode is the eating of Nibor by Diane after Nibor gets down from Clyde.

```
* ?[Diane eat ?x]
# props looked at 1.  # generaliztions 0.
[Diane eat<t3,t4> Nibor]

* ?[Nibor ?y{tp.location} ?z]
# props looked at 1.  # generaliztions 0.
[Nibor on-top-of<t1,t2> Clyde]

* ?s [t2 ?y{tp.abstract-rel} t3]
# props looked at 1.  # generaliztions 0.
[t2 less-than t3]
```

Not all of Nibor's parts have been explicitly talked about so some do not exist in the knowledge base. The next query shows the response of the system when it knows that a dependent concept probably exists, but it has no explicit knowledge about it.

```
* ?s m[?part ?p{tp.part-of} Nibor]
# props looked at 7.  # generaliztions 4.
[Nhead head-of Nibor]

?part is bound to concept c.3
c.3 is dependent on higher-animal which is a generalization
of Nibor
A corresponding concept dependent on Nibor was not found
[?part neck-of Nibor]

[Nbreast breast-of Nibor]
[Nhead head-of Nibor]
```

A story similar to "Little Red Riding Hood", execpt in this one the girl gets eaten by the wolf. First, the actors in the story are extracted by finding the creatures and the humans in the story. Then some of the action in the form of time dependent relationships is found.

```
* ?s m[LRRH is-a-story-in-which [?x ?y{tp.gen:creature}]]
```

Find the possible creatures in the story.

```
# props looked at 1.  # generaliztions 0.
[LRRH is-a-story-in-which [c.29 wolf]].  Need to show: [LRRH
is-a-story-in-which [c.29 creature]]


* ?s m[LRRH is-a-story-in-which [?x ?y{tp.gen:human}]]
```

Find the humans in the story.

```
# props looked at 3.  # generaliztions 0.
[LRRH is-a-story-in-which [c.28 girl]].  Need to show: [LRRH
is-a-story-in-which [c.28 human]]
[LRRH is-a-story-in-which [c.26 woman]].  Need to show:
[LRRH is-a-story-in-which [c.26 human]]
[LRRH is-a-story-in-which [c.27 man]].  Need to show: [LRRH
is-a-story-in-which [c.27 human]]

* ?s m[LRRH is-a-story-in-which [?other ?x{tp.phys-rel}
c.28]]
```

Find all physical relationships with the girl as second argument.

```
# props looked at 1.  # generaliztions 0.
[LRRH is-a-story-in-which [c.29 eat<t7,t8> c.28]]

* ?s m[LRRH is-a-story-in-which [c.27 ?x{tp.phys-rel} c.29]]
```

Find all physical relationships between the man and the wolf.

```
# props looked at 0.  # generaliztions 0.
[LRRH is-a-story-in-which [c.27 kills<t9,t10> c.29]]
```

One other type of sub-net forming predicates are the propositional attitude predicates. The next example looks at some of John's attitudes with respect to animals.

```
* ?s m[John ?att{tp.prop_att} [?x ?y{tp.gen:mammal}]
?belief]
```

Find the entities that may be mammals in John's subnet.

```
# props looked at 2.  # generaliztions 0.
[John thinks [Tom elephant] 1.0].  Need to show: [John
thinks [Tom mammal] 1.0]
[John believes [Spot whale] .99].  Need to show: [John
believes [Spot mammal] .99]

* ?[John ?att{tp.prop_att} [Spot mammal] ?f]
```

Is Spot or one of his generalizations in John's subnet predicated to be a mammal?

```
# props looked at 2.  # generaliztions 2.
Nothing found

* ?s m[John ?att{tp.prop_att} [Spot ?x{tp.gen}] ?m]
```

What are the generalizations of Spot in John's subnet?

```
# props looked at 2.  # generaliztions 2.
[John believes [Spot fish] c.34]
[John believes [Spot whale] .99]
```

In the "real world" Spot is a mammal, but in John's world he has not been predicated directly nor indirectly as one.

```
* ?[Spot mammal]
# props looked at 2.  # generaliztions 1.
Yes
```

The next commands look at the structure of some virtual concepts in John's subnet.

```
* p s John c elephant
Imag15103 of Conc Ext elephant
Indicator links
tp.gen
Access Skeleton
  top_of_hierarchy
    tp.phys-qual
      tp.size Prop17267
      tp.col Prop17353
    tp.gen Prop13954


Uni List
Conc15112 Uni c.30


* p s John conc Tom
Imag16468 of Conc Ext Tom
Function table
  Index f.15
  Objek Imag16656 of Conc Ext Tneck
  Index f.16
  Objek Imag16632 of Conc Ext Thead

Access Skeleton
  top_of_hierarchy
    tp.has-part Prop16589 Prop16606
    tp.gen Prop11468
```